МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева»

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

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель –
кандидат технических наук, доцент
Золотарев В.В.

Красноярск – 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

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. Анализ БДУ ФСТЭК [1] показывает, что доля уязвимостей данного класса с критическим уровнем опасности составляет 33 %, а с высоким – 49 %. Столь высокая оценка опасности связана с возможностью проводить атаки, находясь за периметром информационной системы, реализуя угрозы нарушения информационной безопасности в открытых компьютерных сетях, включая Интернет.

Основной архитектурой в настольном сегменте [2] и серверном [3] является *AMD*64 (и ее аналог *Intel* 64). Выбор ее в качестве рассматриваемой позволяет сделать предлагаемые решения применимыми к ее области использования. Реализуемая в таких ЭВМ Гарвардская архитектура (пример реализации [4, 5]) 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*) [6]. 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* (пример реализации для приложе-

ний – [7], для ядер ОС – [8], особенности и недостатки приведены в [9, 10]), контроль целостности метаданных кучи, различные системы защиты, интегрируемые в ПО (примеры приведены в главе 1). 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 защит. Последнее связано с тем, что при возможности атакующего локально или удаленно взаимодействовать с защищаемым приложением и использовании слабых мест конкретного средства защиты уязвимость может быть реализована.

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. Примером серьезности последствий уязвимостей данного типа является резонансная атака на организацию, разрабатывающую программные продукты *SolarWind* [11], где дефект серверного приложения привел к компрометации инфраструктуры организации. 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 *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 согласно [12, 13], 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.

Как и любая система защиты, предлагаемый метод создает накладные расходы. При этом в зависимости от модели угроз и модели нарушителя существует возможность использовать дифференцированный объем защитных преобразований, обеспечивающий выбор приемлемого соотношения производительности/количества, пригодных для формирования эксплойтов участков.

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 – в силу организационных или технических ограничений. Наличие исходных кодов позволяет гарантировать корректность применения системы защиты. При их отсутствии необходима проверка корректности модификации либо по набору тестов, задающих критерий работоспособности, либо посредством проверок, формируемых самим программным средством, реализующим функции интеграции системы защиты.

Проблемой защиты от *RoP*-уязвимостей активно занимаются в следующих университетах: Флориды (Д. Саливан, 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. Курмангале-

ев). 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 исполнимого кода:

– размещение участков программ случайным с точки зрения 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, так как количество гаджетов не меняется вследствие применения системы защиты.

Целью работы является обеспечение защищенности программ за счет снижения количества пригодных для реализации *RoP*-уязвимостей их 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 неразличимых замен фрагментов кода программы, которые могут быть использованы для проведения *RoP*-атак, использующий свободные ресурсы процессора.

5. Разработать алгоритм расчета метрик защищенности, позволяющий 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 результатами других авторов, проводящих исследования в этой области, а также использованием для перекрестной проверки корректности анализа программ и повышения защищенности к *RoP*-атакам независимых программных средств, являющихся стандартом де-факто.

Научная новизна состоит из предложенных в работе:

1. Модифицированный метод снижения числа пригодных для проведения *RoP*-атак участков в программах, отличающийся от аналогов встраиванием кода системы защиты в программные модули без требования наличия исходных тек-

стов и с обеспечением неразличимости алгоритма защищенной и оригинальной программы.

2. Модифицированная методика контроля целостности графа потока управления, отличающаяся от аналогов использованием псевдослучайных значений для контроля целостности адресов возврата и защитой фреймов стека вызывающих подпрограмм.

3. Модифицированный метод оценки эффективности систем защиты программ от *RoP*-атак, отличающийся 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, обеспечивающего 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 [14-24], 7 из которых изданы в журналах, рекомендованных ВАК [14–20]. Публикации [21–23] 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.

1. Уязвимости возвратно-ориентированного программирования и подходы борьбы с ними

1.1. Введение в проблематику, основные термины и определения

Анализ вопроса появления ошибок в ПО (например, обзор проблемы дан в [28]) позволяет заключить, что конструктивным подходом является не вопрос о том, как исключить их появление, а как минимизировать последствия. Основными направлениями в зависимости от этапа жизненного цикла ПО являются следующие:

– на этапе разработки – путем более качественного тестирования ([29]) или применения средств защиты;

– на этапе эксплуатации – путем исправления дефектов посредством внесения изменений в программный код.

Рассматриваемый вид компьютерных атак [30] основан на комбинировании недостатков программы (уязвимости [31]) для передачи управления на цепочку гаджетов. Эта цепочка обеспечивает выполнение необходимого атакующему кода.

Гаджетами в работе будем называть фрагменты секции кода программных модулей, размещаемых в памяти с правом на исполнение и используемых атакующим в нештатном порядке вызова для реализации алгоритма, не заложенного изначально. Указанные методы в качестве обязательного условия требуют стабильности содержимого модулей, хотя и не требуют для них фиксированной базы.

Пример эксплуатации *RoP*-узвимости показан на Рисунке 1.1. Цифрами в красных кругах отмечены этапы атаки. Повреждение стека может быть вызвано, например, переполнением буфера [32], а адреса гаджетов определяются при атаках на механизм *ASLR* (методы обхода приведены в [33–37]).

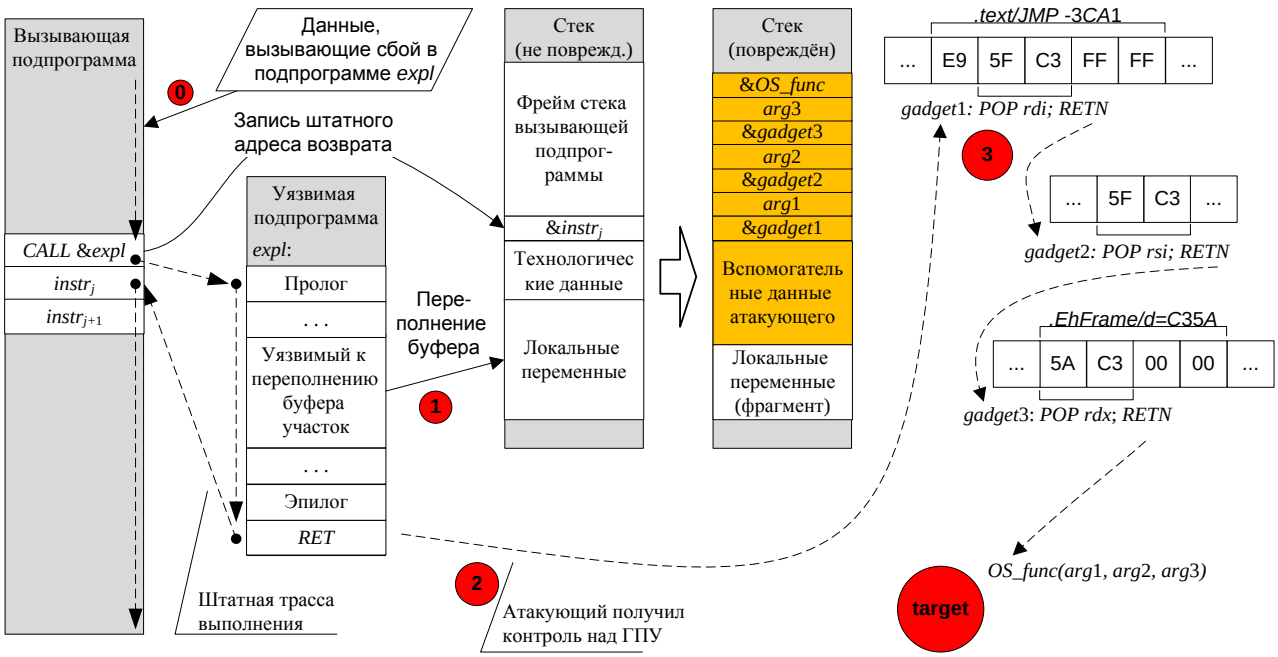

Рисунок 1.1. Пример эксплуатации RoP-уязвимости

В общем виде программы состоят из набора подпрограмм, которые в ходе выполнения вызывают другие подпрограммы. Структура подпрограмм и порядок обмена данными между ними (передача аргументов и получение результата выполнения) определяются бинарным интерфейсом приложений (далее – БИП). Типовая структура подпрограмм включает пролог, основную часть и эпилог. В прологе обычно сохраняются регистры, которые не должны быть повреждены при вызове (в типовом случае – *rbp*), устанавливается база фрейма стека для данной подпрограммы (в регистр *rbp* сохраняется значение регистра *rsp*) и выделяется память под локальные переменные (любые из перечисленных операций могут отсутствовать). В основной части выполняется целевое содержимое подпрограммы (при этом баланс работы со стеком должен быть нулевым – сколько было помещено, столько должно быть извлечено). В эпилоге выполняются в противоположном порядке обратные операции, содержащиеся в прологе. В завершение вызывается инструкция возврата из подпрограммы.

При рассмотрении содержимого стека любого из потоков программы в произвольный момент работы он будет состоять из последовательно записанных фреймов. Пронумеруем их, начиная с нулевого, который соответствует подпро-

грамме, с которой началось исполнение (точка входа, указанная в метаданных программы). Введем обозначения для подпрограмм π_d и адресов возврата ρ_d, где d – некое неотрицательное число. Нижний индекс показывает положение фрейма стека относительно фрейма начальной подпрограммы. Например, если у текущей рассматриваемой подпрограммы фрейм d, то у вызывающей подпрограммы будет фрейм d-1, а у вызываемой – d+1. Описываемые структурные элементы приведены на Рисунке 1.2.

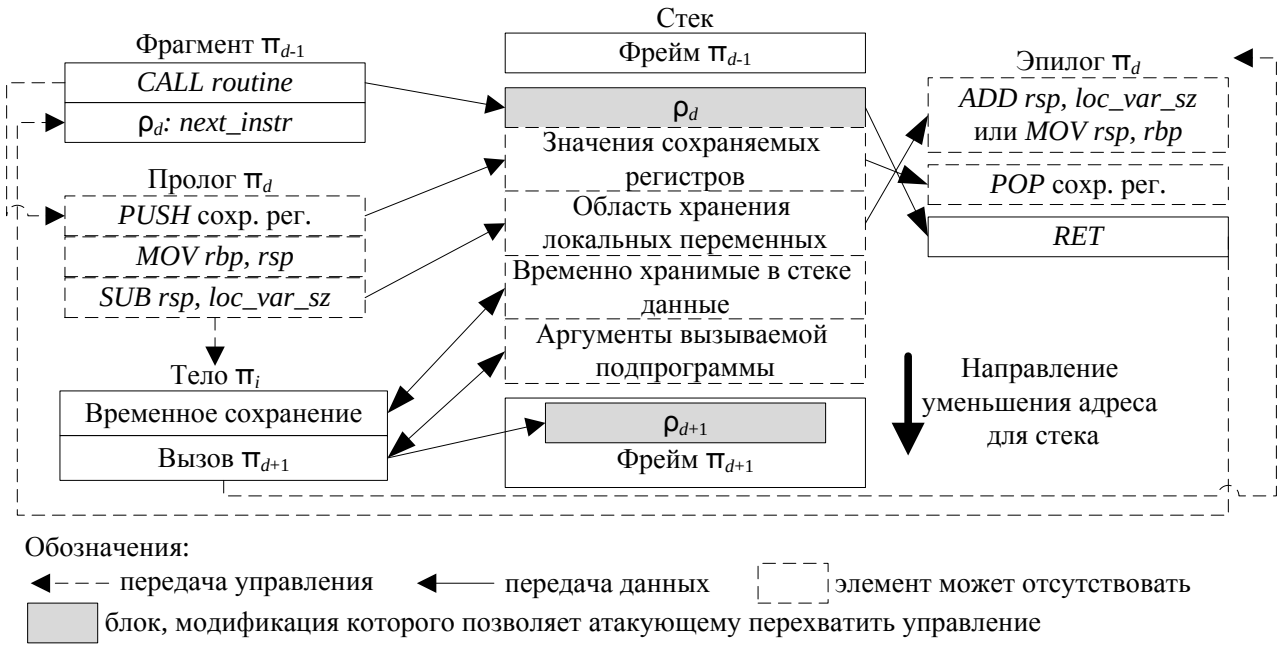

Рисунок 1.2. Типовая структура подпрограмм и их фреймов стека

Для упрощения описания введем понятие опасной инструкции (далее – ОИ). Под ОИ будем понимать инструкции передачи управления по аргументу, который считывается из ОЗУ или регистров процессора. Примерами ОИ являются инструкции возврата *RET* или перехода по адресу, содержащегося в регистре *JMP* <регистр>. Инструкции перехода по фиксированному смещению не относятся к ОИ. Дополнительно введем понятие опасного значения (далее – ОЗ): это последовательность байтов в составе инструкции, которая не является ОИ, при передаче управления на которые они будут интерпретированы процессором как ОИ.

1.2. Анализ уязвимостей удаленного исполнения кода

Для определения объекта защиты выделим критерии применимости участков программ в качестве гаджетов, которые могут использоваться злоумышленником и подлежат защите. Для этого будут использованы данные о структуре подпрограмм [38] и классификации гаджетов из [39].

Для перекрестной проверки была проанализирована выборка уязвимостей (сама выборка и полученные данные приведены в приложении 1). Собираемая информация об уязвимостях и эксплойтах:

– идентификация уязвимости (для однозначной идентификации анализируемого объекта);

– уязвимое ПО, его версия, аппаратное и программное окружение (необходимо для формирования метрик защищенности на реальных объектах атак);

– при наличии опубликованного эксплойта – программные примитивы (гаджеты), используемые для определения состава заведомо уязвимых элементов алгоритма программ, гарантированно требующих защиты;

– закрыт ли исходный код уязвимого ПО (для обоснования актуальности работы системы защиты в ситуации, когда организация, эксплуатирующая уязвимое ПО, не имеет возможности самостоятельно исправлять уязвимости);

– время между публикацией данных об уязвимости и ее исправлением (для уточнения актуальности оперативного блокирования уязвимостей до принятия мер разработчиком);

– при доступности уязвимой версии ПО и эксплойта – сопоставление используемых гаджетов с составом, найденным средствами обнаружения гаджетов (для контроля корректности методического аппарата работы).

Примечание 1. Для отработки предлагаемых решений могут быть рассмотрены программы, в которых синтетически сформированы уязвимости. При этом указанные программы недопустимо использовать для контроля корректности

преобразования алгоритмов ПО вследствие примитивности выполняемых ими задач (например, лишь выводят сообщения об успешности применения уязвимостей).

Примечание 2. Для контроля корректности преобразования алгоритмов ПО целесообразно использовать программы, снабженные средствами контроля функционирования. Такие средства могут быть представлены встроенными средствами самотестирования или модульными тестами, присутствующими в составе исходных кодов СПО.

Ключевые слова, используемые в рамках поиска:

– *return oriented programming;*

– *code reuse;*

– *exploits.*

Рассматриваемые перечни уязвимостей для перепроверки сроков опубликования:

– https://cve.mitre.org

– https://www.exploit-db.com

– БДУ ФСТЭК

Из анализа рассмотренной выборки *RoP*-уязвимостей и эксплойтов для них можно сделать следующие выводы:

– тенденцией является то, что после опубликования наличия уязвимости в ПО или фактического, но не афишируемого существования эксплойта до момента устранения уязвимости может пройти существенное время, в течение которого система может быть эффективно атакована злоумышленником;

– существенную долю уязвимого ПО (24 из 30, три синтетические уязвимости не учитывались) составляют программы с закрытым исходым кодом, что делает невозможным закрытие уязвимости средствами организаций, не являющихся разработчиками;

– следствием указанного выше является актуальность системы защиты, не требующей наличия исходных текстов;

– среди программ, в которых находятся уязвимости, присутствуют версии, для которых, например, не могут быть применены новейшие средства защиты.

В результате были сформированы критерии гаджетов, которые представляют угрозу для защищенности приложений:

– последней инструкцией гаджета является ОИ или ОЗ, интерпретируемая как ОИ. В рамках рассматриваемой архитектуры *AMD*64 ОИ включают инструкции возврата в пределах сегмента (0x*C*3, 0x*C*2) и инструкции возврата за пределы сегмента (двухбайтовые последовательности «0x48 0x*CA*» и «0x48 0x*CB*»). ОИ необходима для передачи управления на следующий гаджет. Адрес передачи управления должен быть предсказуем для атакующего;

– перед последней инструкцией гаджет должен содержать инструкции, модифицирующие содержимое регистров или памяти, эффект выполнения которых необходим для достижения целей атакующего;

– гаджет не должен содержать данных, которые интерпретируются процессором как некорректные инструкции или повреждают данные, необходимые атакующему.

Невыполнение любого из перечисленных критериев делает гаджет непригодным для использования в рамках атаки. Сформулируем модель атаки для определения целей атакующего. Атака осуществляется путем обмена данными с уязвимым приложением локально или удаленно. Модель включает в себя следующие действия атакующего:

– запись при выполнении тела π_d в стек массива данных такого размера, чтобы как минимум произошло повреждение ρ_d. В результате этого он будет заменен на необходимый атакующему адрес ρ_{gadget}, который передаст управление на первый гаджет в цепочке. Повреждение становится возможным при условии некорректной обработки поступающей информации атакуемым приложением. Дополнительно в стек по положительному смещению от места хранения адреса возврата записываются данные для дальней-

ших этапов эксплуатации уязвимости (адреса следующих гаджетов и их аргументы). Примем, что у атакующего в ходе выполнения π_d нет возможности модификации ρ_{d-1} … ρ_0 без повреждения ρ_d вследствие последовательной записи в стек. Из невыполнения данного условия следует наличие у атакующего примитива записи по произвольному адресу, что не позволяет противодействовать такой атаке из-за возможности произвольным образом задавать ГПУ (например, внося коррекции в таблицы виртуальных функций);

– по завершении выполнения эпилога уязвимой π_d выполняется инструкция *RET*, которая передает управление по адресу ρ_{gadget}, записанному атакующим вместо штатного адреса возврата ρ_d. Гаджеты должны заканчиваться инструкцией *RET*, которая считывает адрес следующего гаджета и передает на него управление (саму цепочку формирует атакующий перед атакой);

– выполнение цепочки гаджетов, которое имеет конечной целью вызов функций ОС, которые необходимы атакующему для нарушения безопасности. Для этого атакующему необходимо записать аргументы функции в регистры, используемые для передачи аргументов (определяются БИП), и передать управление на её точку входа из последнего гаджета;

– делается предположение об отсутствии у атакующего возможности контролировать значения произвольных регистров и наличии возможности передавать управление по произвольному адресу. Невыполнение данного предположения делает *RoP*-атаку ненужной и позволяет, например, сделать область стека исполнимой и передать на нее управление посредством классической атаки на переполнение буфера;

– делается предположение о возможности у атакующего сформировать в стеке массив данных, содержащий адреса необходимых гаджетов, и передать управление на первый гаджет не только через повреждение ρ_d. Например, путем записи в единичный регистр, значение которого контролирует атакующий, адреса гаджета и последующего выполнения инструкции ви-

да *JMP* <регистр> (*JoP*-атака). Другим примером является повреждение данных в куче (например, адреса таблицы виртуальных функций с последующим вызовом метода, для которого была выполнена подмена адреса);

– делается предположение о наличии в программе и доступности для атакующего уязвимости, реализующей примитив чтения. Подавая на вход некорректные данные, атакующий сможет получить фрагменты адресного пространства атакуемой программы. Предположим, что за раз считывается фрагмент, соизмеримый с размером фрейма подпрограммы. Примитив чтения позволяет до эксплуатации *RoP*-уязвимости сформировать дамп содержимого исполнимой памяти программы и определить адреса содержащихся в ней гаджетов;

– принимается (с учетом редкости уязвимостей ПО), что в рамках одного вызова подпрограммы не могут быть последовательно реализованы и уязвимость примитива чтения, и уязвимость повреждения ρ_d. Ситуация, при которой в рамках одной подпрограммы атакующий либо читает участок памяти ограниченного размера, либо в рамках нее же повреждает стек, принимается как актуальная.

Использование такой модели позволяет априорно сделать предлагаемые меры устойчивыми к большему числу сценариев атак. Безопасность решения не основывается на неизвестности атакующему сведений о программе или невозможности влиять на управление иначе, кроме как путем повреждения стека. Построенная на таких исходных посылках защита не станет уязвима из-за наличия у атакующего любого примитива чтения.

1.3. Обзор и сравнение существующих подходов по защите от уязвимостей

Проанализируем существующие решения по защите от *RoP*-атак для формирования требований к разрабатываемой системе защиты. Они направлены на

устранение условий проведения атаки или её затруднение. Согласно [39], таковыми являются:

– перехват управления атакующим в результате повреждения адреса возврата в стеке (парируется методом защиты от перехвата управления);

– передача управления на цепочку гаджетов, адрес размещения которых известен атакующему (парируется методом снижения числа пригодных для проведения *RoP*-атак участков в программах);

– известность размещения или наличие дополнительной уязвимости, которая позволяет читать фрагменты адресного пространства атакуемой программы (примитив чтения), в частности для определения адресов гаджетов (парируется методом разового или периодического случайного перемещения фрагментов программы).

Определим критерии сравнения средств защиты для определения их слабых мест и формулирования требований к создаваемому методу защиты. Критерии формируются на основе модели атак (приведенной в подразделе 1.2 и сформированной на основе выборки реальных уязвимостей) и области применимости. В качестве критериев предлагаются:

– в защищаемой системе могут присутствовать как программы с доступным эксплуатанту исходным кодом, так и без такового. Вследствие этого требование наличия исходного кода и проведения перекомпиляции должно быть включено в критерии сравнения, так как оно влияет на применимость средства защиты;

– если средство защиты включает предотвращение перехвата контроля над ГПУ злоумышленником, то должны быть учтены условия, в которых атака может быть проведена (чем они будут сложнее и маловероятнее, тем лучше);

– если средство защиты обеспечивает снижение числа гаджетов, то должны быть учтены условия уязвимости;

– вследствие широкой распространенности узвимостей примитивов чтения, должно быть проанализировано, как её наличие влияет на защищенность.

В Таблице 1.1 приведены ближайшие обнаруженные аналоги, направленные на противодействие *RoP*-уязвимостям. Из анализа приведенной подборки существующих решений противодействия *RoP*-атакам можно заключить, что первым проработанным направлением являются аппаратные системы защиты, но они требуют новейшего или специфичного оборудования, поддержки ОС и представлены не на всех платформах. Они неприменимы для эксплуатируемых систем, аппаратное обеспечение которых не подвергается модернизации (в том числе по экономическим причинам). Вторым распространенным направлением являются встраиваемые на этапе компиляции системы защиты, но они не могут быть применены при отсутствии исходных текстов ПО. Рассмотренные решения, которые применимы в описанной ситуации (*Shuffler* и *RuntimeASLR*), уязвимы к получению атакующим информации о содержимом памяти, так как не ставят своей целью устранение гаджетов, а лишь пытаются сделать место их размещения неизвестным.

Из существующих методов защиты наиболее эффективным является метод снижения числа пригодных для проведения *RoP*-атак участков в программах. Он устойчив к знанию атакующим информации о памяти программы. Предлагается модифицировать данный метод, так как существующие аналоги встраиваются на этапе компиляции. Модифицированный метод защиты должен обеспечивать встраивание кода системы защиты в программные модули без требования наличия исходных текстов и с обеспечением неразличимости алгоритма защищенной и оригинальной программы.

Таблица 1.1. Анализ средств противодействия *RoP*-уязвимостям

Наименование решения	Необх. перекомпиляции	Защита от перехвата управления	Снижение числа гаджетов	Уязвимо при раскрытии информации	Примечание
Control-flow Enforcement Technology (*CET*) [40, 41]	Да	За счет теневого стека	Не требуется за счет контроля над ГПУ	Нет	Поддерживается с 2020 г. [42], но не всеми процессорами не всех изготовителей
PaXgrsecurity RAP [43] и схожее по принципу средство *StackGuard* [44]	Да	За счет контроля целостности адреса возврата	Нет	Да [45, 46]	—
ИСП Обфускатор [47, *48*]	Да	За счет косвенной адресации	Нет	Да [49]	—
Selfrando (и схожие решения [50, 51])	Да	Нет	За счет случайного размещения при запуске	Да [52]	—
Shuffler [53]	Нет	За счет косвенной адресации	За счет периодического случайного перемещения	Да, до момента следующего перемещения	Уменьшение периода негативно влияет на производительность
RuntimeASLR [54]	Нет	Нет	За счет случайного переразмещения в дочерних процессах	Да	—
G-free [55] и работа [56]	Да	Да, за счет контроля целостности адреса возврата	Да, кроме технологических участков и определяемых при связывании адресов	Да, но только в части контроля перехвата управления	—
Scylla [57]	Да	Нет	Да, за счет шифрования участков кода и случайного перемещения	Да, в рамках расшифрованного участка	—

Разрабатываемое средство защиты на основе модифицированного метода защиты должно быть применимо для архитектуры процессоров *AMD*64 (и ее аналога *Intel*64), так как именно для них актуальны *RoP*-уязвимости, что подтверждается рассмотренными источниками. Сформулируем требования к создаваемому средству защиты от *RoP*-уязвимостей:

– исходные коды защищаемой программы не требуются;

– преобразования защищаемого приложения не должны влиять на логику его работы (т.е. на результаты работы приложения);

– должна быть обеспечена невозможность использования штатных инструкций возврата из подпрограмм в качестве гаджетов. То есть после применения мер защиты при замене ρ_d на ρ_{gadget} в рамках выполнения тела π_d недопустим переход по адресу ρ_{gadget} при выполнении штатной инструкции возврата в рамках эпилога;

– должны быть устранены ОЗ. То есть инструкция, в составе данных которой присутствует ОЗ, должна быть заменена на одну или более инструкций, байтовое представление которых не содержит ОЗ.

Предложенные требования формировались для выборки реальных уязвимостей, основанных на технологии *RoP*. При этом этапы эксплуатации узявимости, описанные ранее в рамках модели атаки, частично могут пересекаться с другими методами атак. Отметим, что некоторые из них существуют в виде «доказательств концепции» или теоретических построений. Хотя работа сфокусирована на *RoP*-уязвимостях, рассмотрим влияние реализации изложенных требований на осуществимость других методов атак (с кратким изложением способа атаки):

– атаки, связанные с получением контроля над ГПУ путем повреждения ρ_d (*BROP* [58], *SeBROP* [59], *PIROP* [60], *SROP* [61]), блокируются при реализации предложенных мер защиты для программы и её библиотек;

– реализации Тьюринг-полных операций при обработке данных (*DOP* [62, 63], *BOP* [64], *FOP* [65]) – с одной стороны, реализация предложенных требований не устраняет возможность проведения атаки, с другой стороны, для реализации удаленного исполнения кода для такого вида атаки требует-

ся наличие модифицируемой исполнимой памяти, что не является штатной ситуацией при эксплуатации;

– использование перезаписываемых относительных технологических указателей для преодоления *ASLR* и вызова библиотечной функции с заданными атакующим аргументами (*LOP* [66]) – предложенные в данной работе меры не улучшают и не ухудшают подверженность приложения указанному виду атак. Отметим, что данный вид атак является теоретическим;

– атаки за счет построения гаджетов не на основе инструкции *RET* (*JOP* [67], *Tini-JOP* [68], *PCOP* [69]). Анализ реальных эксплойтов (приведен в Приложении 1) показал, что использование, например, технологии *JOP* не встречается в чистом виде, без комбинирования с *ROP*-гаджетами. В таких случаях предлагаемые меры устранят часть необходимых для успешных атак гаджетов. Остальные технологии являются теоретическими, что затрудняет анализ их практической реализуемости;

– повреждение адресов таблиц виртуальных функций в объектах с построением Тьюринг-полного набора операций на основе диспетчеризируемого вызова методов целиком (*COOP* [70, 71]) – предложенные в данной работе меры не улучшают и не ухудщают подверженность приложения указанному виду атак. Отметим, что для данного вида атаки не удалось найти ни одного реального эксплойта или уязвимости, что позволяет предположить её теоретический характер.

Для реализации предлагаемого средства защиты необходимо разработать алгоритм резервирования в программе участков для вставки кода системы защиты без нарушения ее алгоритма и без использования исходных текстов.

1.4. Проблематика встраивания систем защиты

Встраивание кода в существующее приложение является трудной задачей. Это связано с решением двух проблем: восстановление ссылок между сущест-

вующими частями программы и обеспечение неизменности алгоритма после вставки. Иллюстрация первой проблемы представлена на Рисунке 1.3. Требующие коррекций участки выделены красным, место вставки выделено оранжевым. Все ссылки по абсолютному адресу после места вставки перестают быть корректными. То же касается ссылок по относительному адресу, если база и адрес назначения находятся по смещению с разным знаком относительно места вставки.

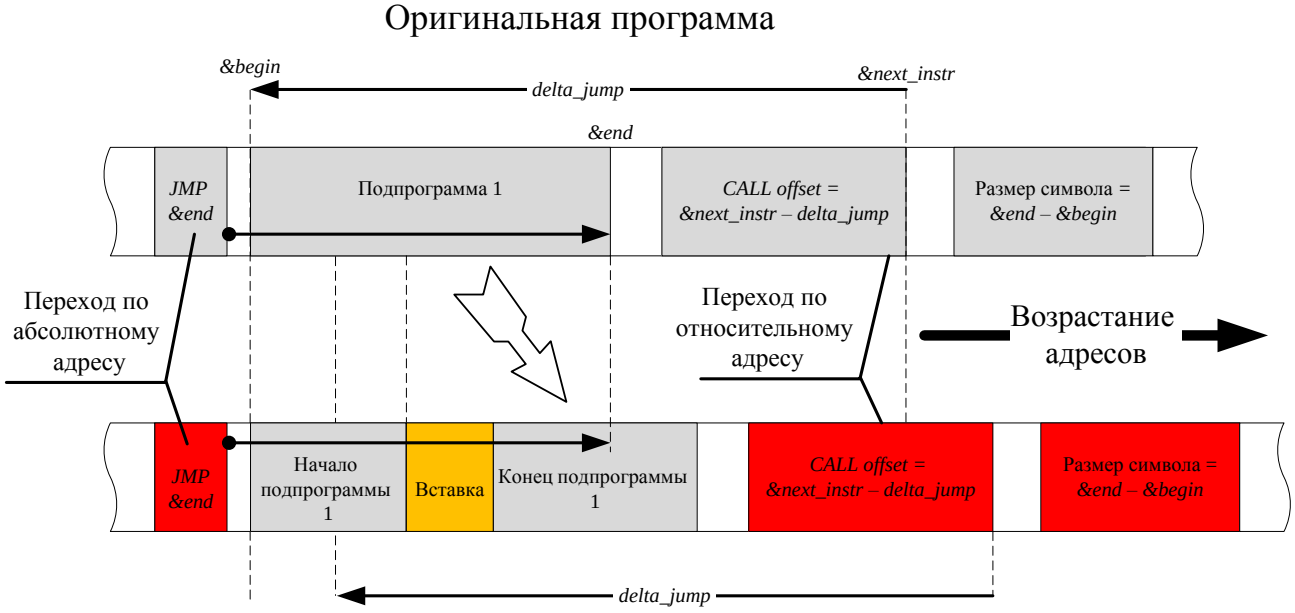

Рисунок 1.3. Возникающие при вставке проблемы адресации

Рассмотрим существующие подходы к встраиванию кода системы защиты, способы обеспечения неизменности алгоритма и особенности образов исполнимых файлов, которые должны быть учтены при решении данной задачи.

1.4.1. Обзор существующих подходов по встраиванию систем защиты

Далее рассмотрены существующие методы модификации программного обеспечения для повышения защищенности от *RoP*-атак в части способов внесения кода защиты.

При наличии исходных кодов защищаемого приложения появляется возможность при каждой компиляции формировать исполнимый файл, содержимое которого является уникальным [48]. Развитием указанного подхода становится внесение динамических изменений в соответствующим образом подготовленный

файл в ходе запуска. Само перемещение участков может выполнять защищаемое приложение [52] либо внешний компонент на основе ассоциированных с образом программы метаданных [50]. При этом фактически происходит подобие повторной сборки приложения.

Наличие исходных кодов защищаемого приложения также позволяет производить модификацию в процессе функционирования. Для этого приложение разбивается на независимые и перемещаемые участки исполнения (оверлеи) [72, 51] или модификация производится внешним относительно программы алгоритмом, размещаемым в ядре [73], или дополнительном потоке [53].

В случае отсутствия исходных текстов (что относится к ситуациям защиты проприетарного ПО), а также нецелесообразности доступа к метаданным времени компиляции применяется дизассемблирование защищаемого модуля с последующим перемешиванием участков. Примером такого подхода является [74]. Недостаток – эмпирический подход к внесению модификаций и отсутствие доказательства корректности анализа.

Отдельно стоит рассмотреть технологии встраивания ВПО в исполнимые файлы в рамках заражения. Примеры описаны в [75]. Спецификой такого встраивания является то, что вопрос сохранения алгоритма исходного приложения не является критичным, так как встраивание производится в рамках скрытия от антивирусных программ. Это приводит к тому, что данные методы неприменимы в рамках систем защиты.

Обобщая, можно сказать, что опубликованные подходы либо требуют наличия исходных кодов и гарантируют корректность, либо не обеспечивают модификацию уже существующих программ и не предоставляют гарантий корректности. Это требует создания решений, обеспечивающих встраивания разрабатываемой системы защиты в программный код обрабатываемого приложения без нарушения алгоритма его работы.

1.4.2. Формат модифицируемых исполнимых файлов

Приведенные в данном подразделе данные получены путем анализа формата исполнимых файлов *elf* [76], стандартов бинарного интерфейса приложений в

Linux [38], заголовочных файлов библиотеки *libelf* [77] и личных исследований автора. Программа может быть условно разбита на три части:

– метаданные (проецируются в память при запуске не в полном составе, но корректность должна быть обеспечена для полного состава);

– данные;

– исполнимый код.

Метаданные подразделяются на:

– заголовок программы – содержит информацию для нахождения адреса, с которого должна начать работать программа, а также для считывания массивов сегментов и секций, которые будут описаны дальше;

– массив сегментов, параметры которых задают сопоставление файловых смещений с виртуальными адресами в ОЗУ для проецируемых фрагментов запускаемой программы, а также атрибуты страниц, используемых для размещения содержимого образа программы;

– массив секций программы – определяет семантику регионов образа программы и описан в Таблице 1.2; идентификаторы согласно перечню типов секций из заголовочных файлов библиотеки *libelf*.

Для всех перечисленных в таблице секций, кроме секций кода и данных, присутствует описание их формата, которое позволяет детерминированно извлекать информацию о ссылках из них, а также делать частичные выводы о структуре секций кода и данных. Например, секция *SYMTAB* может содержать достаточную информацию для разбиения исполнимого кода на подпрограммы, если из образа таковая информация не удалялась.

Для секций, не содержащих исполнимый код и данные программы, исчерпывающая информация может быть получена путем обработки содержимого с учетом формата. Для секций исполнимого кода анализ может быть основан на перечне ссылок внутрь секции кода из других секций, формат которых известен. Дальше должен следовать рекуррентный анализ с учетом особенностей архитектуры процессора, для которого написан исполнимый код (приведены в следующем подразделе). В ходе анализа необходимо выявлять ссылки на другие секции.

Наконец, для секции данных анализ должен быть основан на базе ссылок из других секций.

Таблица 1.2 – Перечень секций и особенности их обработки

Типовое наименование	Идентификатор типа	Назначение	Наличие ссылочной информации
.interp	Не важен	Указание на используемый интерпретатор для запуска	Отсутствуют
Не важно	*NOTE*	Различные поясняющие данные (например, о версии компилятора)	Отсутствуют
*.gnu.** *.comment*	Не важен		
Не важно	*GNU_HASH*	Информация для поиска экспортируемых функций	Отсутствуют
Не важно	*DYNSYM SYMTAB*	Имена, места размещения и, опционально, размеры для участков кода и данных программы (переменных, подпрограмм)	Адреса размещения, размеры
Не важно	*STRTAB*	Список строк для указания имен участков кода и данных	Отсутствуют
Не важно	*DYNAMIC*	Перечень параметров образа программы	Присутствует для конкретных параметров (как адреса, так и размеры)
Не важно	*RELA*	Информация о перечне коррекций, которые нужно внести в программу для изменения базы размещения. Необходимы для работы технологии *ASLR*	Перечень адресов участков памяти. Для части указывается, какой новый адрес должен быть записан в ячейку
.init, .plt .plt.got .text, .fini	Не важен	Исполнимый код программы	Присутствуют, но могут быть извлечены только в при дизассемблировании
.rodata .data.rel.ro .data	Не важен	Данные программы	Присутствуют, но могут быть извлечены только путем анализа семантики исполнимого кода на предмет семантики
.eh_frame_hdr .eh_frame .gcc_except_table	Не важен	Данные для обработки исключений, возникающих в программе [78, 79]	Содержат как абсолютные, так и относительные ссылки друг на друга и на секции исполнимого кода
.init_array .fini_array .jcr, .got, .got.plt	Не важен	Перечень подпрограмм, которые необходимо вызывать на различных этапах жизненного цикла программы	Массивы адресов
.bss	Не важен	Псевдосекция, реально выделяемая после запуска	Отсутствуют
*.debug_**	Не важен	Информация о структуре программы, используемая для целей отладки	Содержат как абсолютные, так и относительные ссылки друг на друга и на секции кода и данных

1.4.3. Особенности архитектуры AMD64

Приведенные в настоящем подразделе данные получены путем анализа описания архитектуры *AMD*64 [80]. Наличия одного указанного источника достаточно, так как алгоритмическая основа вычислений не требует расширений процессора, специфичных для изготовителя. Вследствие этого для восстановления алгоритмической целостности программ достаточно базового описания.

В составе инструкций процессора можно выделить три набора:

– инструкции общего назначения (подлежат анализу, должны декодироваться для анализа образа программы и восстановления адресов);

– системные инструкции (игнорируются в рамках данной работы, так как они не используются в коде штатных программ из состава ОС);

– инструкции расширений процессора (должны анализироваться аналогично инструкциям общего назначения).

Особенности архитектуры, которые должны быть учтены при анализе секций исполнимого кода:

– технология *NX-bit*, которая однозначно позволяет определить, не содержит ли участок программы исполнимый код по результату анализа атрибута исполнимости для сегментов программы;

– *RIP*-относительная адресация, позволяющая указывать положение данных относительно адресующей инструкции – и тем самым снижать накладные расходы на реалокацию и упрощать выявление ссылок на секции данных;

– инструкции условных и безусловных переходов, позволяющие однозначно идентифицировать целевой адрес перехода как начало некоторой инструкции;

– *RIP*-относительная адресация устраняет необходимость указания абсолютных адресов в составе инструкций, а оставшиеся должны быть описаны в рамках секций, ответственных за реалокацию.

В рамках системы команд была выделена специфика инструкций, которая должна быть учтена в ходе анализа данных. Полученные данные сведены в Таблице 1.3.

Таблица 1.3 – Специфика инструкций, ключевых для анализа

Код операции, шестнадцатеричн.	Мнемоника	Признак конца ББ	Вид ссылки	Примечание
0x00	ADD	Нет	—	Выравнивание в секции данных
0x0*F* + (с 0x1*A* по 0x1*F*)	NOP	Нет	—	Выравнивание в секции кода инструкциями, которые не меняют регистры процессора
0x90				
0x0*F* + 0x0*B*	UD2	Нет	—	Заведомо некорректная инструкция
0x0*F* + 0x*B*9	UD1			
0x0*F* + 0x*FF*	UD0			
0x*D*6	—			
0xFF + Mod_rm.reg=3	CALL far			
0xFF + Mod_rm.reg = 5	JMP far			
0x*EA*	JMP far			
С 0x70 по 0x7*F*	Jcc imm8	Да	Относительный, 1 байт	Инструкции условного перехода
0x*E*3	JCX			
0x0*F* + (с 0x1*A* по 0x1*F*)	Jcc	Да	Относительный, 4 байта	Инструкции условного перехода
0x*E*8	CALL	Да, признак начала подпрограммы	Относительный, 4 байта	Передача управления подпрограмме
0x*E*9	JMP	Да, без возможности возврата		
0x*EB*	JMP		Относительный, 1 байт	
0x*C*2	RETN	Да, признак окончания подпрограммы	—	—
0x*C*3	RET			
0x*F*4	HLT	Да, признак окончания программы	—	—
0xFF + Mod_rm.reg = 2	CALL reg/mem	Да, признак начала подпрограммы	Абсолютный, регистр или 4 байта	—
0x*FF* + Mod_rm.reg = 4	JMP reg/mem	Да, без возможности возврата	Абсолютный, регистр или 4 байта	—

1.5. Выводы

В данной главе была рассмотрена проблема *RoP*-уязвимостей в машинном коде и существующих подходов по защите от них. В главе проанализирована выборка из уязвимостей соответствующего типа, показано отсутствие решения данной проблемы даже с появлением технологии аппаратной защиты *CET*, разработанной *Intel*. Для выборки уязвимостей проанализирован состав используемых в эксплойтах гаджетов, статистика инструкций, входящих в их вычислительную часть, и виды управляющих инструкций (инструкции возврата, безусловные переходы и вызовы подпрограмм для аргумента-регистра).

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

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

3) затруднение получения контроля над ГПУ, также выполняемое на этапе компиляции.

Из анализа можно сделать 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 *AMD*64, в качестве операционной системы – *GNU/Linux*, а в качестве бинарного интерфейса приложений – «*System V Application Binary Interface. AMD*64 *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.

2. Резервирование мест для встраивания кода средства защиты при отсутствии исходных текстов программ

Область применимости создаваемого средства защиты не должна быть сужена вследствие исключения программ, исходные тексты которых недоступны. Это является кардинальным отличием предложенного модифицированного метода снижения числа пригодных для проведения *RoP*-атак участков в программах. Сами методы приведения ОИ и ОЗ в непригодный для атак вид приведены в главе 3. В данной же главе описаны решения, позволяющие встраивать предлагаемую систему защиты в программы без наличия их исходных текстов.

В главе проведен анализ структуры программ, их ГПУ, введены определения и сформулирована модель вычисления выходных данных программ. Далее определён критерий неразличимости программ, который показывает условия неизменности алгоритма вычислений, реализуемого в программах. На основании этих результатов сформирован алгоритм резервирования мест для встраивания кода средств защиты, который необходим для замены ОИ и ОЗ на алгоритмически неразличимый код защиты.

2.1. Структура программ и необходимые определения

Ограничим область применения модели вычисления выходных данных программ, так как она не может быть создана универсальной. Среди ряда причин выделим невозможность в общем случае восстановить семантику, заложенную разработчиком в программу. В результате из области рассмотрения будут исключены экзотические случаи, не встречающиеся для подавляющего большинства приложений.

В части ОЗУ рассматривается исключительно обычная память, то есть значения в ней изменяются только вследствие функционирования ядер процессора. Вне рассмотрения оказываются диапазоны адресов ввода-вывода [81], служебные диапазоны адресов, на которые спроецирована память устройств. К указанным

регионам памяти в типовом случае имеет доступ код ядра ОС, для которого изложенная далее модель и алгоритм не применимы. Это не приводит к значимому сужению области применения модели. Аспекты атомарности операций, многопоточности и прерываний не рассматриваются вследствие того, что вопросы синхронизации должны быть решены ее разработчиком с учетом семантики программы, что позволяет рассматривать потоки программы изолированно, как самостоятельные и независимые с точки зрения их алгоритмов. Аналогично не рассматривается обращение защищаемой программы к невалидным диапазонам адресов виртуальной памяти, так как в рамках работы не учитывается корректность защищаемой (оригинальной) программы. Из рассмотрения также исключаются системные регистры, так как их модификация не производится в ходе штатной работы системы и ведет в общем случае к смене режима работы процессора и/или недетерминированному изменению содержимого ячеек памяти.

В качестве основной процессорной архитектуры рассматривается *AMD*64 по причине ее распространенности на момент выполнения работы. Отметим, что в ней присутствует полнота набора команд по Тьюрингу. Разрабатываемые решения актуальны также для набора команд *IA*-32, выполняющихся на процессорах с архитектурой *AMD*64, так как он является подмножеством команд указанной архитектуры. Для архитектур, на которых отсутствует защита от исполнения данных (например, *i*386, *MIPS*), предлагаемые решения ограниченно актуальны, так как при возможности переполнения буфера в стеке программы проще исполнять там же расположенный код, чем формировать *RoP*-цепочки. Для архитектур, реализующих теневой стек (например, *ARM*), предложенные решения нецелесообразны вследствие неприменимости *RoP*-атак.

Не рассматриваются по причине отсутствия или крайней редкости в рамках штатной эксплуатации программных средств:

– самомодифицирующиеся программы (которые недопустимы с точки зрения системы защиты от исполнения данных);

– программы, содержащие вручную написанные ассемблерные вставки (вследствие возможного содержания в них мер противодействия дизассемблированию или анализу);

– ситуации отладки, когда возможна модификация контекста исполнения и данных в адресном пространстве программ внешней программой (в основном отладчиком);

– модели сегментации адресного пространства, отличные от плоской в части сегментов кода и данных (в силу отсутствия фактического применения в рамках операционных систем);

– реализации в рассматриваемом коде обработки прерываний (приводят к неявным переходам, зависящим от метаданных процессора и его режима работы);

– реализации виртуальных машин и интерпретаторов, которые выполняют байткод, или команды, которые могут читать или модифицировать произвольные данные в регистрах или адресном пространстве процесса (в силу того, что атакующий получает фактический контроль над вычислениями в такой ситуации);

– приложения с содержащимися в них программными закладками (в силу возможности внедрения такого вредоносного кода, который позволит обойти произвольную систему защиты вследствие отсутствия механизмов разграничения доступа к регионам памяти в рамках одного адресного пространства).

Хотя приведенный список содержит ряд исключений, указанные ситуации не являются распространенными либо противоречат типовым политикам безопасности, что не ведет к значимому сокращению области применимости. Таким образом, рассматриваются программы, написанные на компилируемых языках программирования (например, Си и Си++), с байтовой структурой любой используемой памяти, каждая ячейка которой содержит 8 бит. Пусть E – множество значений байт, принимает значения от 0 до 255.

Полная информация о программе с учетом перечисленных выше ограничений определяется данными в совокупной памяти (далее – СП), состоящей из ОЗУ и регистровой памяти процессора. Состояние 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 $M^i = (m_0^i, m_1^i, \dots, m_{|M|-1}^i)$, $m_l \in E$, $0 \le l < |M|$ 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.

В ряде ситуаций целесообразно учитывать специфику ячеек СП. В частности, только для ячеек ОЗУ возможна арифметическая адресация, то есть группа значений ячеек может определять адрес другой ячейки ОЗУ. С каждой конкретной ячейкой ОЗУ m_l может быть сопоставлен ее адрес a. Множество допустимых (валидных) адресов на шаге i обозначим как A^i. Сопоставление задается функцией *addr*, определяемой метаданными программы и параметрами работы загрузчика из состава ОС. 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. Введение такого абстрактного индекса и абстрагирование от конкретных адресов необходимо, потому что адреса могут меняться от запуска к запуску. В то же время использовать исключительно индексы нельзя, потому что в рассматриваемой архитектуре применяется технология страничной адресации. Вследствие этого ячейки из M с соседними индексами могут иметь в общем случае произвольные, не соседние адреса. Между такими ячейками ОЗУ могут располагаться регионы недействительных адресов, которым не соответствуют какие-либо ячейки ОЗУ.

Разобьем всё M на множество регионов ячеек (далее – РЯ) ОЗУ так, чтобы в каждом регионе если адреса двух ячеек различаются на единицу, то и индексы их отличались бы на единицу (далее – смежные). Способ разбиения определяется структурой конкретной программы и будет уточняться далее. РЯ 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 зна-

чение v_j^i посредством функции $val{:}M_j^i \to v_j^i, val(M_j^i) = \prod_{k=0}^{n_v-1} m_{j+k} \times 256^k$, где j – индекс первой ячейки в регионе памяти (здесь и далее). Сопоставления регистровой памяти значению могут быть записаны аналогично. При этом конкретный способ сопоставления индексов ячеек регистровой памяти её реальным ячейкам должен быть взаимнооднозначен, но не существенен для рассматриваемого метода. Разбиение на регионы регистровой памяти не нарушает смежность, однако в этом случае под смежностью нужно понимать возможность совместного (в рамках инструкции) обращения к региону (например, ко всем восьми байтам регистра *rax*).

Из-за особенностей архитектуры *AMD*64 перед взаимодействием с какой-либо ячейкой ОЗУ должен быть выработан её адрес. Вследствие этого в процессе работы программы в составе СП должны содержаться адреса ячеек ОЗУ. При этом обратное неверно. То есть если $val(M_j^i) \in A^i$, то не обязательно $val(M_j^i)$ является адресом и обрабатывается как адрес. Этим обусловлена сложность решаемой задачи, так как требуется учет семантики содержимого программы для дифференциации адресов от совпадающих с ними значений. Данная задача не решаема в общем случае, но с учетом введенных ранее ограничений может быть решена автоматически при дополнительных условиях, описанных далее. При дальнейшем описании если указано, что $val(M_j^i) \in A^i$, то указанное значение программой используется в качестве адреса. То есть важно не численное совпадение с каким-либо адресом, но обработка и использование инструкциями в качестве адреса. Введем операцию разыменования: если $a_j = addr(M_j^i)$, то $[a_j]$ тождественно M_j^i.

В составе СП среди ячеек регистровой памяти необходимо особо выделить регион R_{IP}, который содержит адрес исполняемой в данный момент инструкции программы. Именно посредством его модификаций выполняемыми инструкциями осуществляются условные и безусловные переходы.

В составе M могут быть выделены следующие подмножества: исполнимый код O, который штатно не может быть изменен в ходе выполнения программы, и данные программы D (как изменяемые, так и неизменяемые, включая служебные

данные и непосредственные аргументы инструкций). Такое разбиение отражает принципиально разные подходы к анализу их содержимого (формат исполнимого кода однозначно определяется архитектурой системы, а формат данных зависит от реализованных в *O* алгоритмов и ОС). Информация о размещении сведена в Таблице 2.1.

Таблица 2.1

Подмножество *M*	Местоположение на этапе анализа	Местоположение на этапе исполнения
Исполнимый код анализируемой программы *O*	Базовый адрес размещения содержится в метаданных программы. Если код не является позиционно-независимым, то в составе программы содержатся абсолютные адреса, принадлежащие данному подмножеству. Иначе используются относительные адреса, рассчитанные с учетом размещения секций в памяти	Адрес точки входа содержится в регистре R_{IP} на момент передачи управления программе
Данные программы из состава секции кода, входящие в *D*		Если используются абсолютные адреса, тогда выполняется либо загрузка по фиксированному в метаданных адресу, либо в метаданных программы должна присутствовать таблица реалокации. Возможно определение местоположения по относительным адресам, содержащимся в секции кода или данных
Инициализированные данные из состава секции данных, входящие в *D*		
Неинициализированные данные из состава секции неинициализированных данных, входящие в *D*		
Стек	Не определено. Относительное позиционирование косвенно задается фреймами подпрограмм, позиция фрейма определяется значением в регистре R_{SP} на момент начала ее выполнения	Начальное значение определяется при передаче управления на точку входа значением R_{SP}, для конкретной подпрограммы значение однозначно определяется совокупностью фреймов последовательности вызванных подпрограмм
Иные модули	Не определено. Может быть штатно получено только посредством функций ОС	Не определено. Может быть штатно получено только посредством функций ОС
Служебные структуры данных		

Для удобства выделим из состава СП множество регистровой памяти *R* и данных в ОЗУ *D*, которые включают как модифицируемую память, так и данные, размещаемые совместно с исполнимым кодом (аргументы операций) и доступные только для чтения. Содержимое множества *D* определяется однозначно: в него входит вся неисполняемая память (этот атрибут следует из метаданных программ), аргументы операций (следуют из формата для архитектуры команд) и

байты выравнивания (недоступные для передачи управления фрагменты программы между участками кода). Будем называть такой массив состоянием системы S^i размером $|S|$. Это допустимо сделать, так как мы исключили из рассмотрения самомодифицирующиеся программы, а также тот факт, что память, содержащая исполнимый код, штатно не модифицируется им же самим. Состояние вычислений на шаге i может быть представлено как

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

Часть ячеек из состава D не имеет абсолютных адресов на этапе анализа (в частности, фреймы подпрограмм, размещаемые в стеке), вследствие чего они должны рассматриваться как смещения относительно начальных состояний соответствующих ячеек из состава R^0 (т.е. значений в регистровой памяти на момент запуска программы). При этом на этапе исполнения у всех ячеек памяти адрес будет уже конкретным. Все рассматриваемые регистры имеют исключительно абсолютную идентификацию и не поддерживают относительного указания.

Множество O содержит данные, которые в ходе штатного функционирования программы интерпретируются процессором как набор команд (операций, инструкций). С точки зрения детерминированности результата используемые в программе инструкции могут быть разделены на два класса: результат работы которых зависит только от входных данных, и второй класс – дополнительно зависящие от внутреннего состояния процессора или устройств, недоступных программе. Результат выполнения инструкций первого класса может быть представлен 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). Таким образом, программу можно представить как набор инструкций $f_{j\,i}$, являющихся чистыми функциями [82, стр. 581]. Они не зависят от внутреннего состояния процессора, результат их работы определяется исключительно содержимым СП на момент исполнения. Инструкции $f_{j\,i}$ могут повторяться для некоторых i. Подавляющее число инструкций общего назначения входят в данный класс.

Рассмотрим второй класс инструкций. В него включены:

– специальные инструкции общего назначения (например, *RDRAND, RDPID*);

– инструкции передачи управления на системные библиотеки, алгоритм которых не определен на этапе анализа защищаемого приложения и которые могут выполнять системные вызовы к ядру;

– системные инструкции, разрешенные в рассматриваемой ОС для вызова непривилегированным приложением (например, *RDTSC, SYSCALL*);

– привилегированные в рассматриваемой ОС системные инструкции – *INT, OUT, STI* (исключаются из дальнейшего рассмотрения вследствие неприменимости разрабатываемого средства к коду ядра ОС).

Результат выполнения инструкций второго класса может быть 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 (например, получение пользовательского ввода, текущего времени, состояния операционной системы), т.е. зависит от шага i выполнения программы в общем случае.

Для удобства описания ГПУ и сторонних эффектов будем представлять код программы не в виде инструкций, а более крупными фрагментами. Согласно критериям полноты по Тьюрингу, программа может быть представлена в виде наборов безусловно и последовательно выполняемых цепочек инструкций и операций условного ветвления между такими цепочками. С учетом специфики построения реальных программ будем использовать понятие базового блока [83, p. 231] инструкций (далее – ББ). Критерием окончания ББ являются: инструкции условных или безусловных переходов, вызовы подпрограмм и специальные инструкции, которые не могут считаться чистыми функциями, хотя в этих ситуациях фактически отсутствует ветвление алгоритма. Критерием начала ББ является либо конец предшествующего ББ, либо передача управления из другого ББ.

При таком подходе программа может быть представлена в виде ориентированного ГПУ. В нем из каждого узла исходит одно (безусловный переход или неявный переход в рамках последовательного выполнения инструкций, не осуществляющих условный или безусловный переход), два ребра (инструкции условного перехода), неопределенное количество (безусловный переход по аргументу при вычислении адреса перехода на основе данных программы; например, в рамках таблиц переходов). Узлам ГПУ соответствует ББ инструкций b_j, состоящий из последовательности инструкций $f_j \ldots f_{j+|bj|-1}$ (где j – индекс первого байта первой инструкции и, как следствие, первого байта ББ, и $|b_j|$ – число инструкций блока), выполняемых в рамках указанного блока, по завершении которого определяется, по какому ребру графа будет осуществлен переход. Указание размера ББ в его обозначении не является обязательным, так как критерий разбиения программы на них однозначно его определяет.

С учетом того, что некоторые ББ могут оканчиваться операцией, не являющейся чистой функцией, запуски программы с одинаковым состоянием S^0 будут приводить к различным S^i для каждого запуска. Для блоков такая 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 $\tilde{b}_{j^i} = \tilde{f}^i_{j^i} \circ b_j$. 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 ББ, оканчивающихся инструкцией условного перехода или перехода по вычисляемому адресу (явно или неявно, как в случае с исключениями языков программирования), выбор следующего ББ зависит от состояния на момент начала ББ, но не состояния на момент его окончания. Вследствие этого трасса выполнения программы не повторяется в общем случае при запуске с одинаковым начальным состоянием, но различными наступившими сторонними эффектами.

2.2. Модель вычисления выходных данных программ

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 программы. Если эти результаты семантически совпадают, то делается вывод об идентичности программ.

Программа в начальном состоянии может быть представлена как

$$P = R^0 \cup D^0 \cup B$$

$$B = \left(\dots, b_j, \dots\right), \forall j$$

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

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

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

Количество шагов i в общем случае неопределенно (и может стремиться к сколь угодно большим значениям), равно как и последовательность фактически исполненных инструкций процессора в ходе выполнения программы, согласно теореме об остановке Машины Тьюринга [84, стр. 230].

Алгоритм работы программы определяет порядок преобразования входных данных и инициализированных данных из состава образа программы во выходные значения. Входные данные поступают через регистры и память и представляют собой модификацию содержимого подмножества ячеек S^i в результате $\tilde{f}^i_{j\,i}$. Выходные данные выдаются аналогичным путём, но вместо модификации происходит чтение. С учетом ранее описанной особенности работы с памятью для использования ячеек как входных или выходных их адрес должен использоваться в рамках $\tilde{f}^i_{j\,i}$. При этом используемые в ходе такой операции ячейки, хранящие адрес, не могут рассматриваться как содержащие выходное значение. Это обусловлено тем, что их содержимое используется как метаданные лишь для указания на целевые

для операции ячейки. Для регистров, наоборот, используется статичное определение задействуемых ячеек. Состав ячеек, используемых в качестве входных или выходных, в общем случае не известен, но для конкретных случаев может быть определен при существовании протоколов обмена программы с библиотеками или ядром ОС. Во избежание сужения области применимости разработанной модели будем считать, что все ячейки, не содержащие метаданные, являются входными и выходными данными одновременно. 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 в формализованном виде:

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

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

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 аргументов $S^i_{jp} \subset S^i_p$ с точки зрения того, содержат ли они адреса (без разыменования, что соответствует адресной арифметике или перемещению данных) 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: $val\left(S^i_{jp}\right) \subset A^i, \nexists m^i_k \in S^i_p : m^i_k = \left[val\left(S^i_{jp}\right)\right]$ 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.

Пример выполняемых инструкций программы с выделением входных и выходных данных показан на Рисунке 2.1.

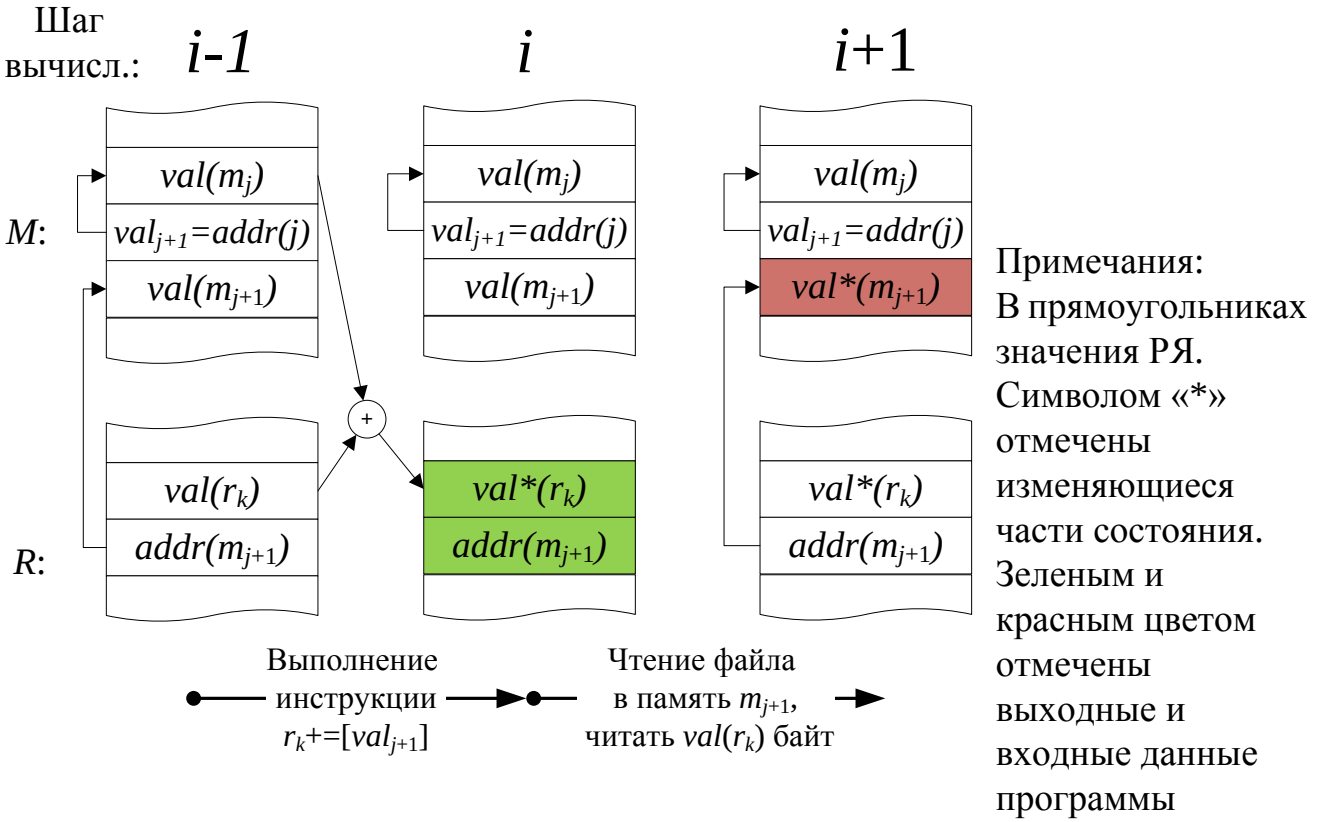

Рисунок 2.1. Входные и выходные данные программы

Основываясь на модели вычислений выходных данных программ, сформулируем условия выполнения критерия неразличимости программ. Рассмотрим простейший случай выполнения неразличимости для двух запусков одной и той же программы. Это позволит определить базовые условия, которые должны выполняться и для пары «оригинальная – модифицированная» программа.

2.3. Условия самонеразличимости программ

При рассмотрении двух запусков одной программы их можно принимать неразличимыми, если состояния поэлементно одинаковы, что автоматически приводит к выполнению критерия самонеразличимости:

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

Так как преобразования $\tilde{f}_j^i$ от запуска к запуску не влияют на корректность их функционирования, то примем в рамках рассмотрения вопроса неразличимости, что сторонние эффекты одинаковы для обеих рассматриваемых программ.

Допустимо принять равенство R^0, так как оно определяется загрузчиком программы и при одинаковом окружении будет одинаково. Также необходимо отметить, что значения в R^0 не могут быть предсказаны в общем случае и должны рассматриваться как неизвестные значения. При этом указанное множество будет включать регистры R^0_{IP} и R^0_{SP}, которые определяют конкретные значения, используемые для относительных переходов между инструкциями и размещения и (или) считывания данных из стека соответственно.

Вследствие того, что рассматривается два запуска одной и той же программы, будем считать для них содержимое кода O и данных D^0 идентичным. С учетом принятых допущений неразличимость программ достигается при функциональной неразличимости их ББ, а сторонние эффекты могут не учитываться (и приниматься одинаковыми для любых запусков P):

$$P \rightleftarrows P' \leftrightarrow \text{для } \forall j: b_j = b_j'$$

Тогда вне зависимости от порядка вызова ББ в ГПУ результат работы программ будет одинаков (то есть две программы будут неразличимы для стороннего наблюдателя). Условия неразличимости неприменимы при любом внесении изменений в b_j, так как, например, при любом изменении их размеров определяемый после выполнения последней инструкции блока адрес следующего ББ R^i_{IP} будет содержать некорректный адрес и переход по нему приведет к нарушению корректности вычислительного процесса в целом.

Любая коррекция ГПУ приведет к нарушению критерию самонеразличимости оригинальной и модифицированной программы. Если мы скорректируем b_j так, чтобы адрес следующего ББ вычислялся корректно, то любой вызов подпрограммы поместит в S'^i значение адреса возврата, равного адресу начала следующего смежного ББ, что приведет к неравенству с S^i. При этом не обязательно будет нарушена семантика вычислений. Под этим понимается то, что адреса в программе не являются целевыми данными для вычисления и служат цели адресации элементов M, используемых для внутренних вычислений. Вследствие изоляции адресных пространств конкретный адрес бесполезен для внешней программы и обладает семантикой только в рамках конкретного процесса. Возможны ситуации, когда в выдаваемых программой данных содержатся адреса (приведены типовые ситуации в качестве примеров):

– при использовании атакующим эксплойта, реализующего примитив чтения (является нештатным функционированием, исключается из рассмотрения);

– при чтении регионов памяти отладчиком (ситуация отладки не является штатной при рассмотрении работы программы в рамках эксплуатации);

– указание адресов регионов памяти при взаимодействии с ОС или иными программами (сами регионы формируются ОС, как и их адреса, вследствие чего учитывать такой вариант не требуется).

Приведенные выше рассуждения показывают, что целевыми результатами вычислений являются не адреса, но данные, расположенные в R и в ячейках D, обладающих адресами, но не содержащих адреса.

2.4. Условия неразличимости программ с учетом семантики

Сформулируем условия неразличимости программ с учетом их семантики. Для этого рассмотрим требования, которыми нужно дополнить условия самоне-

различимости. Для этого предварительно рассмотрим особенности задания адресов ячеек памяти, неразличимости состояний и ГПУ.

Адреса ячеек могут быть заданы в двух формах: относительные и абсолютные. Далее будем использовать термин «ссылочная информация» в качестве обобщающего наименования обеих форм адресов. Примерами являются относительные адреса в инструкциях перехода, абсолютные адреса в таблице экспорта, размеры подпрограмм (как смещение от начала первой и до конца последней инструкции) в секции обработки исключений. Если ячейка памяти не входит в состав f_j как непосредственный операнд, то обращение процессора к ячейкам M всегда требует указания конкретного адреса. Это верно в том числе и при относительной адресации, которая сводится к указанию в подмножестве ячеек M значения смещения, которое прибавляется к неявно заданному абсолютному адресу (например, следующей за текущей инструкции при выполнении операций условных переходов или начала структуры, которая содержит поле с относительным адресом). В символьном виде вычисляемый абсолютный адрес ячейки, задаваемый в относительной форме, может быть представлен как $a_t = a_{base} + \Delta a = a_j^i + v_j^i$, где a_t – целевой адрес, по которому расположены обрабатываемые значения.

Ранее рассмотренная специфика разбиения программы на ББ упрощает процесс коррекции и не требует оперирования с каждым отдельным байтом или инструкцией, что замедлило бы процесс внесения изменений. Целесообразно рассмотреть специфику работы с данными для аналогичного упрощения, не нарушающего семантику программы. Рассмотрение вопроса [85, стр. 463] показало, что множество D^0 формируется компилятором из так называемых «символов», то есть блоков данных (далее – БД), характеристиками которых являются адрес и тип, который определяет размер блока и порядок работы с ним. Адресация БД в момент исполнения программы осуществляется по абсолютным адресам (базовым адресам БД). При этом обращение к данным внутри БД производится по относительным адресам, вычисляемым как базовый адрес плюс смещение до необходимого поля в пределах БД. Фактически структура БД отражается в программном коде, который его обрабатывает. Штатно никакие участки, кроме начала БД, не

адресуются вследствие отсутствия в этом смысла (так как для адресации достаточно относительных адресов). При наличии внутри БД адресов они также указывают на другие БД или ЛБ. При формировании модифицированной программы оптимизированное обращение с БД может быть построено следующим образом: БД считать массив ячеек ОЗУ, начиная с некоторой адресуемой ячейки, и до ячейки, предшествующей следующей адресуемой. У такого подхода имеется недостаток: если в конце блоков имеются байты выравнивания (для того чтобы следующий БД располагался с адресом, выровненным по машинному слову или параграфу), то при их перемещении эти байты могут стать бесполезными, а выравнивание будет утрачено. При передислокациях для исключения сбоев, связанных с нарушением взаимного расположения БД, необходимо использовать принцип: если БД был выровнен, то он остается выровнен после модификации; если БД располагались смежно, то эта смежность не должна быть потеряна.

В рамках вставки кода системы защиты между существующими ячейками осуществляется вставка ячеек, которые заполняются холостыми инструкциями (*NOP*) и не соответствуют каким-либо ячейкам оригинальной программы. Вследствие этого ячейки до и после вставляемого участка перестают быть смежными. Это определяет *addr'* – новый способ сопоставления адресов с ячейками *M'*. Так как производится вставка холостых инструкций, то выполняемые в рамках *B* преобразования останутся неизменными.

Если инструкция $f_{j\,i}$ взаимодействовала (выполняла чтение и/или запись) с ячейкой m_j^i с адресом $a_j^i = addr(m_j^i)$ в оригинальной программе, то аналогичное взаимодействие с той же ячейкой в модифицированной программе, но по новому адресу $a'^i_j = addr'(m_j^i)$ не приведет к изменению в смысловой части. Аналогично и с размещением инструкций самой программы. Результат вычислений будет идентичным, если в ходе вычислений получены те же значения и к ним есть доступ (т.е. если в модифицированной программе в семантически тех же ячейках СП содержатся адреса, которые указывают на одинаковые целевые данные; иными словами, важны не сами адреса, а значения, по ним содержащиеся).

Рассмотрим содержащиеся в памяти объекты в M, равно как и их адреса, с точки зрения неразличимости обработки:

– Содержатся в P изначально (размещаются в памяти загрузчиком и могут модифицироваться им для реалокации). Для обеспечения неразличимости такие адреса должны быть при модификации скорректированы рассмотренным далее образом, а алгоритм их обработки должен совпадать.

– Поступают в программу как результат выполнения библиотечных функций и системных вызовов (не могут быть предугаданы при рассмотрении программы, так как входят в состав сторонних эффектов). Для обеспечения корректности достаточно совпадения алгоритма их обработки. При выполнении условия совпадения сторонних эффектов данные адреса будут численно совпадать. Далее такие объекты могут не рассматриваться.

Рассмотрим условия формирования P' из P, при которых будет выполняться критерий неразличимости программ. Уточним условия выполнения семантической неразличимости состояний программы. Два некоторых состояния неразличимы ($S^i \rightleftarrows S'^i$) при условиях:

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

2. Пусть $\Delta v_j^i : a_j^i = a_{basej}^i + \Delta v_j^i$ – смещение относительно базы a_{basej}^i, тогда для $\forall j$: если $a_j^i \neq a'^i_j$, 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 $v_k^i \in A^i$ и $v_k^i = a_j^i$, 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 эквивалентных 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.

Рассмотрим результат преобразования S^i и S''^i таких, что $S^i \rightleftarrows S'^i$, программой при выполнении одной и той же $f_{j\,i}$. Для этого рассмотрим обработку входных аргументов операции для каждого типа согласно документации на процессор:

1. Обработка значения, не являющегося адресом. Никакие адреса не модифицируются (условия 1, 2 и 3 можно не рассматривать). При выполнении условия 4 вследствие одинаковости обрабатываемого значения такой аргумент окажет одинаковое влияние на следующее состояние при выполнении одинакового f_j. Это обеспечит выполнение условия 4 для рассчитываемых на основе данного значения элементов S^{i+1}.

2. Адреса обрабатываются, но не разыменовываются. Над адресом возможны следующие преобразования:

– Копирование. Замена a^i_j на a''^i_j не меняет семантики вследствие того, что адресуемая ячейка в соответствии с условием 1 остается той же. Это обеспечит выполнение условий 2 или 3 для ячейки S^{i+1}, куда будет помещен a''^i_j.

– Сложение адреса со смещением. Выполняется в рамках относительной адресации. Если адресуется БД, то, согласно условию 5, если результат операции в оригинальной программе лежал в пределах, то и в модифицированной программе ситуация сохранится. Если адресовывался участок кода, то, согласно условию 2, целевая ячейка будет одинакова для оригинальной и модифицированной программы.

– Операции маскирования или считывания младших бит адреса. Например, маскирование разрядов адреса для обеспечения выравнивания адреса. Преобразование может встречаться при работе с выровненными данными (стеком или страницами памяти), но такие адреса с подобной спецификой не содержатся в программе изначально. При нарушении выравнивания программа завершает работу с ошибкой, следовательно, данный механизм не предназначен для изменения значения адреса, а значит, не меняет адресуемую ячейку.

– Прочие операции над адресами. Не имеют смысла по различным причинам. Операции сложения и деления – вследствие того, что результат такой операции будет находиться вне диапазона допустимых адресов и непредсказуем из-за механизма *ASLR*. Битовые операции не над младшими битами – вследствие того, что адреса не имеют битовой структуры в рамках пользовательского режима. Корректность таких преобразований не может быть обеспечена без анализа семантики преобразований ББ, как минимум содержащего рассматриваемую инструкцию, а как максимум – региона программы, на который влияет результат преобразования. Если при анализе F встретились указанные операции над адресами, то неразличимость не может быть обеспечена, а встраивание кода защиты на основании предлагаемого подхода невозможно. Отметим, что указанные операции могут встречаться при добавлении кода систем защиты, что соответствует нарушению условия наличия ассемблерных вставок. Сами компиляторы, согласно приведенной статистике, указанные преобразования над адресами не формируют.

3. Происходит разыменование адреса. Замена a^i_j на a'^i_j не меняет семантики вследствие того, что адресуемая ячейка в соответствии с условием 1 остается той же. Операция будет произведена в модифицированной программе над той же ячейкой, что и в оригинальной.

Таким образом, если $S^i \rightleftarrows S'^i$, то $f_{j\,i}(S^i) \rightleftarrows f_{j\,i}(S'^i)$. Это является шагом индукции. Из этого также следует одинаковость переходов в ГПУ, так как, согласно условию 3, если $v^i_{IP} \in A^i$, $val(v^i_{IP}) \rightleftarrows val(v'^i_{IP})$, а согласно используемому в качестве базового 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 $D^0 \rightleftarrows D'^0$. Рассматриваемые состояния соответствуют содержимому образу программы, размещенному в ОЗУ загрузчиком.

Рассмотрим значения элементов D^0 с точки зрения содержащихся в нем адресов. Для обеспечения неразличимости они должны быть выявлены и скоррек-

тированы при модификации программы рассмотренным далее образом. Для этого необходимо рассмотреть то, как программа хранит и вырабатывает адреса в процессе работы.

Рассмотрим множество D^0 с точки зрения содержащихся в нем адресов и необходимых коррекций при внесении изменений в программу. Сами адреса могут быть как относительными, так и абсолютными. Кроме того, адреса могут указывать как на ячейки множества O (по определению содержат адрес какого-либо $\tilde{b}_{j\,i}$), так и на РЯ из D^0. Вариант записывается в скобках как: (вид адреса, ячейку из какого множества памяти он адресует).

Для адресов, содержащихся в составе операндов f_j инструкций (за счет чего вид адреса однозначно идентифицируется по их формату):

– (абсолютный, O) – при перемещении адресуемого b_j должен быть скорректирован на новое значение;

– (относительный, O) – должен быть скорректирован при изменении взаиморасположения ББ (рассматриваемого и адресуемого). Для случаев относительного перехода на первую инструкцию следующего смежного ЛБ должна быть явно добавлена дополнительная инструкция безусловного перехода по абсолютному адресу следующего ЛБ (преобразование, соответствующее такой инструкции, меняет только R_{IP} и не нарушает, а наоборот, обеспечивает неразличимость состояний, так как именно благодаря ему сохраняется целостность ГПУ и переход происходит по адресу, соответствующей той же ячейке, но в модифицированной программе);

– (абсолютный, D^0) – корректируется при перемещении адресуемой ячейки с данными программы;

– (относительный, D^0) – применяется при реализации позиционно независимого кода, отсчитывается в типовом случае от R_{IP}, который указывает на начало следующей после исполняющейся инструкции, корректируется при изменении взаиморасположения адресуемой ячейки и использующей такой вид адресации $\tilde{f}^{i}_{j\,i}$.

Для адресов, содержащихся в подмножестве ячеек D^0:

– (абсолютный, O) – требует коррекции при изменениях адреса целевого b_j (используется, например, для организации таблиц переходов и таблиц импорта/экспорта). Необходимо отметить, что на ячейки либо на начало структуры данных, содержащей ячейки с адресом данного типа, должна присутствовать ссылка из памяти O для обеспечения их включения в обработку;

– (относительный, O) – должен быть отброшен, так как при операциях с данными относительные адреса в архитектуре *AMD*64 не могут быть использованы в силу отсутствия необходимых инструкций. Использование таких конструкций сделало бы невозможной реалокацию в ВАП программных модулей и применение технологии *ASLR*;

– (абсолютный, D^0) – требует коррекции при изменении расположения адресуемой ячейки;

– (относительный, D^0) – адресация выполняется только в пределах смежных регионов памяти. Для достоверного нахождения и коррекции таких адресов необходим анализ операций, обрабатывающих память с адресами рассматриваемого типа. Это существенно усложняет методику модификации, но может быть обойдено при совместном перемещении смежных регионов D^0 на одинаковое смещение, что снимает необходимость коррекции таких адресов.

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 $\forall i: S^{\mathrm{i}} \rightleftarrows S'^{\mathrm{i}}$, а, следовательно, $P \rightleftarrows P^{'}$.

2.5. Алгоритм резервирования мест для кода средства защиты

Приведенная модель вычисления выходных данных программ и условия неразличимости позволяют сформировать алгоритм резервирования мест для встраивания кода средств защиты с сохранением алгоритма (обеспечением семантической тождественности). Это создает основу для замены опасных с точки зрения *RoP*-программирования участков на не пригодные для использования в рамках уязвимостей. В ходе указанного процесса модифицированные участки в общем случае будут иметь иной размер, чем участки оригинальной программы. Но если в случае модифицированного участка меньшего размера можно заполнить образовавшееся пустое место инструкциями вида *NOP*, которые не выполняют никаких преобразований состояния, кроме модификации R_{IP}, то для случая большего размера необходимо изменить месторасположение всех участков с адресами большими, чем адрес модифицируемого участка.

Алгоритм резервирования мест для встраивания кода средств защиты приведен на Рисунке 2.2. Данный алгоритм позволяет выделить области памяти, в которые будет помещен код защиты, формируемый в соответствии с разделом 3.

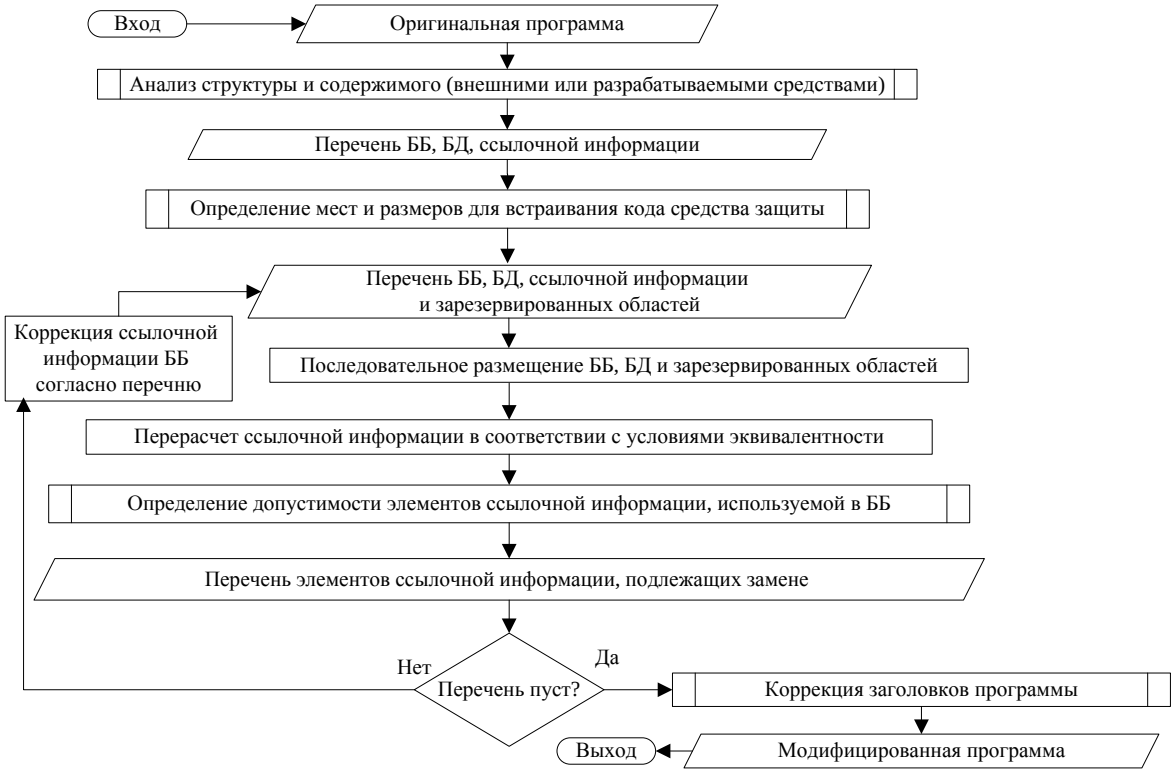

Рисунок 2.2. Алгоритм резервирования мест для встраивания кода средств защиты

Первая операция алгоритма направлена на определение, из каких ББ и БД состоит программа, а также адресных связей между ними. Именно эта операция определяет, будет ли неразличимы алгоритм оригинального приложения и модифицированного. В случае корректного определения этих данных выполнение условий неразличимости обеспечит идентичность выходных данных при одинаковых входных. Используемые средства анализа описаны в подразделе 2.6.

В рамках следующей операции определяется набор мест оригинальной программы, куда осуществляется вставка дополнительного кода (в частном рассматриваемом случае это код защиты). В результате для каждого конкретного опасного участка будет сформирован алгоритмически неразличимый защищенный участок. Порядок определения мест для вставки, формирования алгоритмически неразличимого кода системы защиты и доказательство их неразличимости приведены в разделе 3.

После определения полного состава выполняется итерационный процесс формирования образа модифицированной программы, содержащей код и данные. В рамках каждой итерации выполняется последовательное размещение ББ и БД в памяти. Далее производится коррекция ссылочной информации в соответствии с условиями неразличимости, изложенными в подразделе 2.4. В рамках коррекции производится оценка корректности адресной информации по двум критериям для последующей замены. Во-первых, на вхождение в ОДЗ инструкций передачи управления. Этот процесс гарантированно конечен, так как в пределе произойдет замена всех однобайтовых относительных ссылок на четырехбайтовые. Во-вторых, значения адресов проверяются на наличие ОЗ (критерии приведены в подразделе 1.2, порядок устранения и обоснование сходимости алгоритма содержатся в пункте 3.2.2). Если перечень подлежащей замене ссылочной информации пуст, то выполняется заполнение метаданных программы, в результате чего формируется готовый к исполнению образ. Если перечень не пуст, то производится коррекция ссылочной информации и выполняется следующая итерация.

2.6. Средства анализа исполнимых файлов

Выделение участков в обрабатываемой программе для размещения кода системы защиты требует знания фактического размещения элементов программы в ее исполнимом образе. При этом важна достоверность определения, так как неверная интерпретация, например, адреса не позволит его скорректировать. Для получения сведений о структуре исполнимых файлов могут как применяться существующие дизассемблеры, так и разрабатываться в рамках данной работы вспомогательные решения. Рассмотрим различные варианты формирования сведений о ББ, БД и адресах в рамках решаемой задачи.

Дизассемблер *IDA*. Данное программное средство является стандартом дефакто в области анализа исполнимых файлов. Согласно [86], основным функционалом являются (с примечаниями автора о полезности в рамках данной работы):

– распознавание стандартных библиотечных функций (крайне полезно для контроля качества распознавания);

– интерактивность работы (позволяет настраивать формат отображения листинга, эмулируя пользовательский ввод);

– развитая система навигации (позволяет эффективно находить проблемный с точки зрения анализа участок программы, эмулируя пользовательский ввод);

– система типов и параметров функций (не имеет практической значимости в рамках данной работы);

– встроенный язык программирования *IDC* (не имеет практической значимости в рамках данной работы);

– открытая и модульная архитектура (если в ходе реализации алгоритма перемещения будет принято решение создавать плагин для повышения эффективности извлечения данных об анализируемой программе, то данный функционал обеспечит это);

– возможность работы практически со всеми популярными процессорами (что создает возможность для расширения перечня поддерживаемых архитектурами процессоров);

– возможность работы практически со всеми популярными форматами файлов (что создает возможность для расширения перечня поддерживаемых форматов для программной реализации методики);

– работа со структурами данных высокого уровня: массивами, структурами, перечисляемыми типами (крайне полезно для контроля качества распознавания).

В рамках непосредственного анализа функционала посредством тестовой эксплуатации было дополнительно выявлено следующее:

– закрытый код в совокупности с отсутствием исчерпывающей информации о логике работы приложения является недостатком, так как при выявлении сбоя анализа нет возможности без исследования проблемы сделать вывод о ее причинах;

– доступна для некоммерческого использования бесплатная версия, поддерживающая анализ приложений для архитектуры *AMD*64;

– не все секции декодируются без дополнительных плагинов (которые не всегда существуют), что не позволяет для них полноценно проанализировать качество работы разрабатываемого средства.

Дизассемблер *Ghidra*. Согласно главной странице информации о продукте [87], ключевыми особенностями являются:

– поддержка платформ: *Windows*, *Mac OS* и *Linux*;

– поддержка сборки, пересборки, визуализации анализируемого программного средства;

– поддержка различных форматов исполнимых файлов и различных архитектур процессоров;

– разработка плагинов и скриптов для автоматизации работы.

В рамках анализа функционала посредством тестовой эксплуатации было дополнительно выявлено следующее:

– решение имеет открытый код, но это не дает преимуществ, так как в случае сбоя анализа диагностическая информация носит недостаточный характер, что приводит к необходимости исследования проблемы;

– лицензионные ограничения, которые не позволяют получить доступ к дистрибутиву;

– в ходе поверхностного аудита средства выявлялись дефекты [88], которые могут быть интерпретированы как НДВ, что делает примеение средства небезопасным для использования до проведения полного аудита.

Прочие средства. Были проанализированы такие средства, как *radare*2 [89], *gdb* [90], *OllyDbg* [91], *WinDbg* [92], и в рамках тестовой эксплуатации они показали меньший функционал, чем ранее рассмотренные средства, что делает их использование нецелесообразным.

Разрабатываемые в рамках работы решения. Вне зависимости от выбора средства анализа в рамках работы необходим декодер инструкций для выявления представления инструкции как набора полей. Далее в структуру, содержащую данные декодированной инструкции, должны быть внесены коррекции для восстановления логической целостности программ. Затем должно быть произведено кодирование инструкции с модификацией содержимого памяти. Вследствие указанных причин необходим программный примитив кодирования и декодирования инструкций и целесообразно, чтобы он являлся логически совмещенным для снижения вероятности сбоев и связного устранения существующих проблем.

Необходимые программные примитивы рассчитаны на одну и ту же архитектуру и имеют однозначный критерий корректности работы (вследствие детерминированности описания архитектуры). Таким образом, за критерий выбора допустимо принять полноту реализации состава инструкций, которые могут быть обработаны. Это упрощает выбор, так как для целевой архитектуры полноту обработки обеспечивает только библиотека *fde*64 [93].

Для обеспечения лицензионной частоты может применяться вынесение указанной библиотеки из состава проекта с обменом исключительно посредством данных.

Выбор средства. В рамках сравнения преимуществ и недостатков возможных решений видится целесообразным следующий состав и зона ответственности применяемых средств:

– для операций кодирования и декодирования, а также восстановления логической структуры программ применяется разрабатываемое в ходе работы средство;

– типовые фрагменты программ (например, секции с известным форматом) должны анализироваться разрабатываемым средством;

– для контроля корректности его работы применяется *IDA free* [94].

2.7. План тестирования реализации алгоритма встраивания

Критическим в рамках данной работы является контроль качества работы предлагаемой методики, так как нарушение неразличимости делает защищенную программу непригодной для эксплуатации. Для обеспечения достоверности получаемых результатов были предприняты следующие меры:

– Формирование синтетических тестовых программ, библиотек и модульных тестов для них, обеспечивающих контроль корректности работы со 100 % покрытием тестами по ветвлению алгоритма и составу исполненных строк. В тестируемые участки программного кода не включаются технологический код, генерируемый компилятором и выполняемый автоматически в рамках запуска и исполнения программ (например, код обработки массивов вызова конструкторов, деструкторов, динамического определения адресов импортируемых функций). Рассмотренные технологии: исключения, процедурные типы, статические объекты и переменные. Проверка проводилась для уровней оптимизации от 0 до 3.

– Сопоставление выявленных блоков (кода, данных) и ссылок с дизассемблером *IDA free* до модификации и сопоставление набора блоков после

модификации с результатами того же дизассемблера после модификации. Состав выявленных блоков и ссылок должен быть не хуже.

– Отработка методики на наборе программ, основанных на различных технологиях программирования и заимствованных из состава ОС. Если в ходе программ выявлялись и вручную верифицировались (в основном по исходному коду) обстоятельства, выходящие за границы применимости методики, то программы исключалась из выборки. Критерием отбора программ для включения в отработочный набор является их использование для функционирования операционной системы и возможность проверки вручную.

– Контрольная группа, отобранная по тем же критериям, что и отработочная.

– Контроль корректности на основании программ, с исходными кодами которых поставляются тесты для проверки их корректности. Примеры программ: *cppcheck* [95], *asn1c* [96].

В рамках контроля корректности модификации в случайные участки каждой определенной подпрограммы в автоматическом режиме вставлялись блоки от 16 до 4096 инструкций *NOP*, а далее проводилось автоматическое восстановление структуры программы по приведенному выше алгоритму. При сбое анализа модификация не производилось и выполнялось выяснение причин сбоев. Критерием сбоя анализа является наличие по завершении анализа блоков, в которых содержится исполнимый код, но в ходе его анализа выявляются некорректные инструкции, что является признаком неверного определения ссылочной структуры программы. При необходимости вносились коррекции в программную реализацию (при выявлении ошибок) или программа исключалась из тестирования (для случаев тестовых выборок).

2.8. Выводы

В главе предложена формальная модель вычисления выходных данных программ и вводится критерий эвивалентности программ. Далее рассмотрены условия неразличимости программ (самонеразличимости и семантической), на основании которых строится алгоритм резервирования мест для встраивания кода средств защиты, а также порядок контроля качества выполняемых преобразований.

В рамках модели вычислений выходных данных программ исследуемое приложение рассматривается как последовательность операций над ячейками памяти. Неизвестное на этапе анализа подмножество ячеек используется для ввода в программу обрабатываемых данных и для вывода результатов обработки данных из программы. Не представляется возможным определить их состав без анализа семантики. Поэтому критерий неразличимости отражает то утверждение, что если после модификации все ячейки с результатами вычислений содержат те же результаты, что и их аналоги в оригинальной программе, то тогда работа двух программ при рассмотрении выходных данных будет неразличима.

Для описания состояния используется множество ячеек с данными (изменяемыми или нет). Инструкции программы рассматриваются как выполняющие преобразования из текущего в следующее. Далее формулируются условия неразличимости состояний с учетом семантики программ, идея которой состоит в одинаковости адресов для ячеек памяти, хранящих семантически одинаковые значения, но не адресной инфомации. Далее показывается, что после выполнении операций над неразличимыми состояниями порождаемое состояние остается неразличимым. Из этого следует, что для неразличимости программ необходимо добиться неразличимости начальных состояний.

Для обеспечения неразличимости начальных состояний проанализированы виды ссылок в программе и их возможное местонахождение. На основании указанной информации предложен алгоритм резервирования мест для встраивания

кода средств защиты, позволяющий выделять в программах места для кода системы защиты с сохранением их неразличимости.

Таким образом, в данной главе сформулирована основа для первого защищаемого положения. Оно отражает особенность модифицированного метода снижения числа пригодных для проведения *RoP*-атак участков в программах, заключающейся в возможности встраивания предлагаемого средства защиты в программы при отсутствии их исходных текстов или при невозможности перекомпиляции.

3. Защита программ от уязвимостей возвратно-ориентированного программирования

В 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 ОИ и ОЗ, а также приводятся детали реализации указанного алгоритма с учетом избегания недостатнов, присущих существующим средствам.

3.1. Алгоритм устранения опасных инструкций и опасных значений

Согласно требованиям к создаваемому средству противодействия *RoP*-атакам, сформулированным в подразделе 1.3, необходимо заменить содержащий гаджеты код оригинального приложения на код, сформированный системой защиты. Этот код должен быть алгоритмически неразличим от соответствующего кода защищаемого приложения.

Проанализируем возможные места размещения ОЗ и эффективные меры по их устранению. В соответствии с подразделом 1.2, такими местами являются:

– код системы защиты, размещаемый в эпилогах подпрограмм. Модификация последовательностей инструкций перед *RET* должна осуществляться таким образом, чтобы исключить чтение со сдвигом. Такое чтение в ходе атаки позволяет интерпретировать фрагменты существующих инструкций как необходимые атакующему другие инструкции. Защита обеспечивается за счет формирования перед *RET* такой последовательности инструкций, которая не может быть интерпретирована атакующим как содержащая полезные для атаки операции;

– фрагменты инструкций, не являющихся ОИ, которые подлежат синонимической замене с сохранением алгоритма оригинальной программы;

– области размещения данных, по технологическим причинам являющиеся исполнимыми и которые должны быть лишены признака исполнимости.

С точки зрения модели вычисления выходных данных программ, ОИ могут содержаться только в составе множества O, а ОЗ могут содержаться как в составе множества O, так и в составе множества D^0. Никакие модификации вследствие выполнения алгоритма резервирования мест для встраивания кода средств защиты не меняют элементы множества O, но модифицируют элементы множества D^0, содержащие ссылочную информацию. Вследствие этого поиск ОЗ в ссылочной информации из состава D^0 нецелесообразен и должен быть выполнен после корректировок ссылочной информации в D'^0 на шаге «Определение допустимости элементов ссылочной информации, используемой в ББ» алгоритма резервирования (описан в подразделе 2.5). Если элемент ссылочной информации модифицированной программы содержит ОЗ, то он признается недопустимым, производится коррекция в соответствии с пунктом 3.2.3 и выполняется повторная попытка размещения. Рассмотрим алгоритм устранения ОИ и ОЗ, приведенный на Рисунке 3.1.

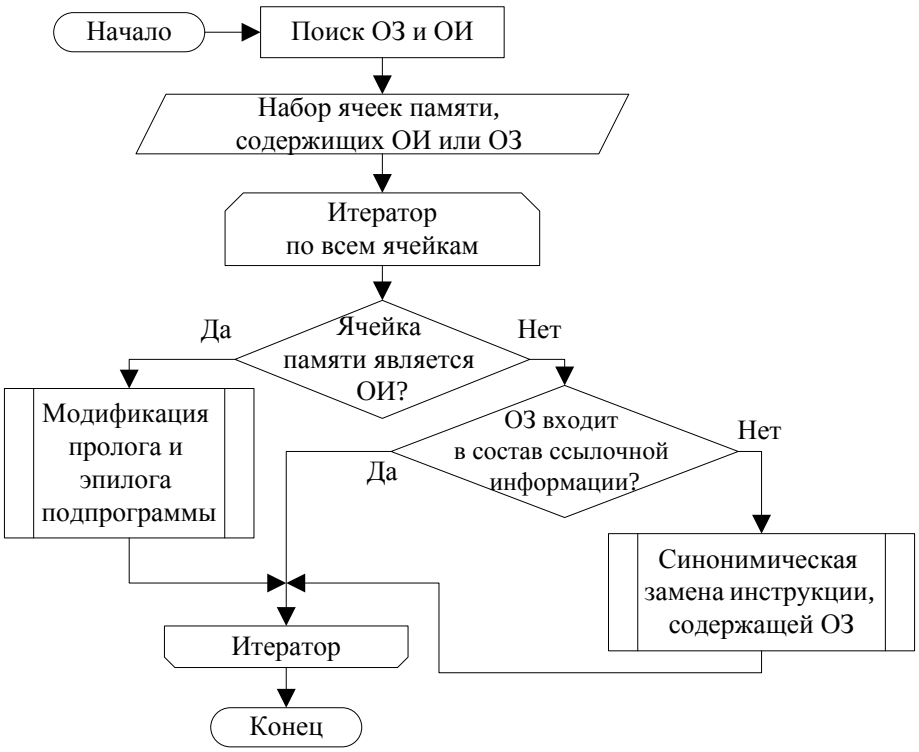

Рисунок 3.1. Алгоритм устранения ОИ и ОЗ

Поиск в коде оригинальной программы ОЗ и ОИ выполняется сравнением значений проверяемых байт с перечнем, определенным в подразделе 1.2. Для решения данной задачи классический алгоритм *GALILEO* (описан в [97]) не применим, так как синонимичные замены или коррекции ссылочной информации могут изменить содержимое перед ОЗ. Это потенциально ведет к тому, что неприменимое в составе гаджета ОЗ из оригинальной программы приведет к появлению гаджета в защищенной программе. Вследствие этого все ОЗ, не являющиеся ОИ, подлежат устранению.

Модификация пролога и эпилога подпрограммы проводится согласно методике контроля целостности графа потока управления, описанной в подразделе 3.2. Синонимическая замена инструкции, содержащей ОЗ, проводится согласно подразделу 3.3.

3.2. Методика контроля целостности графа потока управления

В главе 1 данное направление защиты было сформулировано как одна из составляющих, обеспечивающих защиту штатных инструкций возврата. Из рассмотрения существующих подходов к решению данной проблемы можно выделить два направления: замена ОИ с косвенным указанием ρ_d и контроль целостности ρ_d. Недостатком первого подхода является утрата бинарной совместимости кода со стандартными библиотеками ОС, невозможность использования в качестве функций обратного вызова, невозможность реализации исключений. Второй подход не имеет указанных недостатков, но уязвим к специфичным для метода атакам. Для контроля целостности адреса возврата используется некое значение (условно ключ системы защиты), на основе которого и ρ_d в прологе π_d вычисляется значение, которое будет использовано для контроля целостности в эпилоге. Если атакующий компрометирует ключ (общий для всех подпрограмм), то любая из подпрограмм становится уязвима. Для компрометации достаточно прочитать любой фрейм стека посредством примитива чтения. Для устранения данного недос-

татка предлагается метод защиты, который является развитием второго подхода к защите.

Если для защиты ОИ из состава π_d при каждом запуске формируется непредсказуемое или трудно предсказуемое (для простоты далее называется случайным) значение κ_d, то для возможности подмены ρ_d необходимо получение содержимого фрейма стека, включающего это значение. Техническим ограничением является скорость генерации κ_d. Сравнение возможных источников приведено в Таблице 3.1.

Таблица 3.1. Сравнение возможных источников κ_d

Источник κ_d	Характеристика источника	Кол-во случайных бит	Возможность компрометации	Примечание
Инструкция RDRAND	Декларируется как криптографический стойкий ГПСЧ	32 или 64	Аппаратные закладки или уязвимости	Доступна в процессорах Intel с 2013 года, а производства AMD – с 2015
Инструкция RDTSC	Недетерминированность из-за прерываний, промахов кэша, затрат на синхронизацию, возврата значений для разных ядер и т.п.	Минимум 4 (по результатам экспериментов)	Нет из-за невозможности построения физической модели процессора	Доступна на всех процессорах с архитектурой AMD64
Инструкция AESENC	Стойкость одного раунда шифрования, используется генерируемый RDRAND-ключ	128 или 256	Аппаратные закладки или уязвимости, криптоанализ значений κ_{d-1} XOR κ_d	Доступна в процессорах Intel с 2008 года, а производства AMD – с 2010
ГПСЧ, предоставляемые ОС или в составе эпилогов	Криптографически стойкий	Не менее 32	Нет из-за изоляции процессов в ОС	Имеет неприемлемую скорость генерации
ГПСЧ, функционирующий на выделенном ядре	Криптографически стойкий	Не менее 32	Локальные атаки	Ресурсы одного ядра становятся недоступны
Прочие источники случайности	Отсутствуют гарантии непредсказуемости в общем случае	Не определено	Нет оценки	Например, «мусор» в стеке

Помимо качества случайного значения с точки зрения эксплуатации также важна скорость работы. По результатам сравнения были оценены как перспективные варианты: *RDRAND*, *RDTSC*, *AESENC* и ГПСЧ на отдельном ядре. Длитель-

ность выполнения данных инструкций, определенная экспериментально (измерялась длительность 2^{24} вызовов), составила 730, 35, 4 и 20 (соответствует чтению из памяти). Дальнейшее их сравнение проводилось по результатам апробации. Для снижения накладных расходов защитное преобразование ρ_d также должно занимать минимальные ресурсы. При этом криптографическая стойкость такого преобразования не требуется. Вследствие этого может быть выбрана любая обратимая операция. В рамках рассмотренных существующих решений в подавляющем большинстве случаев используется *XOR*. Данный вариант был также выбран для предлагаемой системы защиты.

Согласно модели атаки, повреждение ρ_d может быть осуществлено в ходе выполнения тела π_d. Вследствие этого защитное преобразование должно быть выполнено до тела, в прологе, а контроль целостности адреса – после тела, в эпилоге. Вследствие того, что при выполнении атаки посредством повреждения ρ_d штатное продолжение функционирования программы будет невозможно в любом случае, то допустимо в защищенной программе аварийно завершать ее функционирование при выявлении попытки получения злоумышленником контроля над ГПУ. Вследствие этого, если при повреждении ρ_d управление будет передано кодом системы защиты на непредсказуемый адрес, то это будет приемлемым решением. Таким образом, для контроля целостности в эпилоге ρ_d заменяется на ρ_d *XOR* κ_d, а в прологе на основании того же ключа за счет повторной операции преобразования восстанавливается корректный адрес возврата. Если атакующий заменит хранимое в стеке значение ρ_d *XOR* κ_d на ρ_{gadget}, то в результате работы кода системы защиты управление будет передано на адрес ρ_{gadget} *XOR* κ_d, что не позволит провести успешную *RoP*-атаку, так как значение κ_d неизвестно атакующему.

Для корректной работы такой системы защиты сгенерированное в прологе значение κ_d должно быть сохранено до эпилога для восстановления адреса. Место хранения должно выбираться так, чтобы оно не было доступно атакующему. Для этого могут быть использованы регистры общего назначения (далее – РОН) или регистры расширения. При этом программа должна быть проанализирована на

наличие гаджетов и штатных инструкций, позволяющих атакующему влиять на место хранения. Подверженные влиянию регистры не должны использоваться для защиты. Отметим, что хранение в стеке κ_d неприемлемо, так как при возможности переполнения буфера атакующий может заменить его значение на нулевое, а ρ_d – на ρ_{gadget}. Тогда управление будет передано на нужный ему адрес.

Проблемой описанного подхода является то, что вследствие ограниченности числа регистров процессора сгенерированное в прологе $\pi_{d\text{-}1}$ значение $\kappa_{d\text{-}1}$ должно где-то храниться на время вызова π_d (и для этого не могут использоваться те же регистры). Решением является сохранение $\kappa_{d\text{-}1}$ в стеке в рамках пролога π_d и возвращение в регистр-хранилище в рамках эпилога.

Место сохранения $\kappa_{d\text{-}1}$ выбирается так, чтобы не нарушать алгоритмическую неразличимость оригинального и защищаемого приложения. Данному условию соответствует единственное место – между областью хранения локальных переменных и временно хранимыми в стеке данными (типовая структура подпрограмм и их фреймов стека приведена в подразделе 1.1). Это обусловлено тем, что данные, начиная с аргументов предшествующей подпрограммы вплоть до области хранения локальных переменных текущего фрейма стека, адресуются относительно базы стека, хранимой в регистре *rbp*. База стека численно равна адресу следующего за адресом возврата хранимого в стеке значения. Временно хранимые в стеке данные адресуются относительно текущего значения регистра *rsp*. Таким образом, любое количество данных, помещенное в предлагаемое место, не влияет и не нарушает логику работы приложений.

Гарантированное завершение выделения места в стеке под локальные переменные происходит к концу пролога. Помещение в стек временно хранимых там данных гарантированно происходит не ранее тела подпрограммы. Вследствие этого единственным допустимым местом для вставки кода системы защиты, которое выполняет сохранение $\kappa_{d\text{-}1}$, является место между прологом и телом подпрограммы. И наоборот, все временно хранимые данные будут удалены из стека до начала выполнения эпилога, а освобождение области хранения локальных переменных начнется не раньше него же. Соответственно, извлечение из стека $\kappa_{d\text{-}1}$

должно быть произведено после окончания тела подпрограммы, но не ранее начала исполнения эпилога.

Хранение κ_{d-1} в непреобразованном виде создает изъян безопасности (описан ранее), вследствие чего необходимо хранить значение в виде «κ_{d-1} *XOR* κ_d» (решение проблемы первого в цепочке значения κ_0 приведено далее). Для реализации данного варианта в ходе встраивания системы защиты должен проводиться анализ неиспользуемых регистров. Статистика встречаемости регистров-аргументов приведена в приложении 3. При вызове внешних подпрограмм текущей подпрограммой ее κ_d либо должно быть сохранено в стеке, либо для вызываемой внешней подпрограммы должно быть известно, что она не модифицирует хранилище κ_d, либо для вызываемой библиотеки должна быть применена описываемая система защиты.

Анализ защищенности приведенного выше решения показал, что при наличии у атакующего примитива чтения в π_l может быть прочитан фрейм стека π_d, где $l > d$. Таким образом, атакующий получит значение ρ_d *XOR* κ_d. Исходя из предсказуемости выполнения трасс программ, атакующий может определить значение ρ_d и вследствие этого получить значение κ_d. В том же фрейме хранится и κ_d в открытом виде. Если в π_d содержится уязвимость переполнения буфера, то атакующий, заменив защищенный адрес возврата на ρ_{gadget} *XOR* κ_d, перехватит управление над выполнением программы.

Для защиты от реализации данного сценария необходимо выполнить в рамках пролога π_d дополнительное защитное преобразование над местом хранения защищенного ρ_{d-1} с использованием κ_d. Также κ_{d-1} в рамках пролога должен помещаться в стек в защищенном виде как κ_{d-1} *XOR* κ_d.

При выполнении π_0 предшествующего фрейма стека не существует. Для решения данной проблемы в рамках кода подготовки программы к запуску генерируется значение κ_{init}, которое сохраняется в стеке и указывается как адрес возврата вызывающей подпрограммы для π_0. В ходе выполнения π_0 хранимое значение κ_{init} будет заменено на κ_{init} *XOR* κ_0. Таким образом, значение κ_{init} не хранится в незащищенном виде и может быть восстановлено атакующим только при знании

$к_0$. Допустимо принять, что ГПУ стабилен и определен для атакующего, поэтому $ρ_d$ известно атакующему (так как вызывающая подпрограмма определяется ГПУ), а, следовательно, он может вычислить «$к_d$ *XOR* $к_{d-1}$» для любого d.

Методика контроля целостности ГПУ реализуется путем вставки кода средства защиты в зарезервированные участки в каждой подпрограмме защищаемого приложения. Запишем в формальном виде преобразования, выполняемые над регистрами и стеком добавляемым кодом средства защиты в прологе $π_d$, которые выполняются в d и d-1 фреймах стека (для $d > 1$):

$$ρ_d \rightarrow ρ_d \, XOR \, к_d$$

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

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

При выполнении пролога $π_{d+1}$ в фрейме d будет выполнена модификация:

$$ρ_d \rightarrow ρ_d \, XOR \, к_d \, XOR \, к_{d+1}$$

В эпилогах будут выполнены 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.

При отсутствии повреждения защищенных адресов возвратов аргумент инструкции *RET* будет аналогичен оригинальной программе и управление будет передано вызывающей подпрограмме. Таким образом, предлагаемая мера защиты не искажает ГПУ.

Покажем в алгебраическом виде неразличимость. Пусть для хранения к используется регистр расширений R_{xmm15}, а регистр расширений R_{xmm14} хранит адрес размещения ключа в предыдущем фрейме стека (указанные регистры не должны использоваться оригинальным приложением). Рассмотрим вызов $π_d$, которая в рамках пролога выделяет в стеке v_d байт. Пусть на момент передачи управления на начало пролога регистры и память имеют следующие значения: $R_{SP} = R_{SPd}$,

$[R_{SPd}] = \rho_d$, $R_{xmm15} = R_{xmm15(d\text{-}1)} = \kappa_{d\text{-}1}$, $R_{xmm14} = R_{xmm14(d\text{-}1)}$, $[R_{xmm14(d\text{-}1)}] = \rho_{d\text{-}1}$ *XOR* $\kappa_{d\text{-}1}$. Тогда на момент завершения пролога единственным регистром из рассматриваемых, у которого изменится значение, будет $R_{SP} = R_{SPd} + v_d$. В ходе выполнения кода системы защиты, размещаемого после пролога, будут выполнены следующие преобразования: $R_{SP} = R_{SPd} + v_d + 16$ (т.е. выделение в стеке 16 байт после области хранения локальных переменных), $[R_{SP}] = \rho_d$ *XOR* κ_d, $R_{xmm15} = \kappa_d$, $R_{xmm14} = R_{SPd}$, $[R_{SPd} + v_d] = \kappa_{d\text{-}1}$ *XOR* κ_d, $[R_{SPd} + v_d + 8] = R_{xmm14(d\text{-}1)}$, $[R_{xmm14(d\text{-}1)}] = \rho_{d\text{-}1}$ *XOR* $\kappa_{d\text{-}1}$ *XOR* κ_d, $R_{xmm15} = \kappa_d$. Если π_d содержит вызов подпрограммы π_{d+1}, то состояния регистров на момент её запуска будут соответствовать аналогичным значениям вызывающей подпрограммы с учетом замены индексов с d-1 на d для регистров расширений и d на $d + 1$ для регистра *rsp*. При выполнении кода системы защиты, размещаемого перед эпилогом, контроля целостности графа потока управления будут выполнены следующие преобразования: $R_{r8} = \kappa_d$, $R_{xmm15} = R_{xmm15(d\text{-}1)} = \kappa_{d\text{-}1}$, $R_{xmm14} = R_{xmm14(d\text{-}1)}$, $R_{SP} = R_{SPd} + v_d$. Данные преобразования не меняют логики работы приложения, так как значения в регистрах *r*8 и *r*9 гарантированно не используются после завершения тела подпрограммы согласно БИП. При этом восстанавливается значение регистра R_{SP} на окончания выполнения пролога. Это позволяет коду в эпилоге корректно освободить локальные переменные и подготовить стек для возврата из подпрограммы. К моменту завершения эпилога регистр R_{SP} содержит адрес возврата из текущей подпрограммы R_{SPd}. Далее выполняется код средства защиты, размещаемый непосредственно перед ОИ в конце подпрограммы. После его выполнения: $[R_{xmm14(d\text{-}1)}] = \rho_{d\text{-}1}$ *XOR* $\kappa_{d\text{-}1}$ *XOR* κ_d *XOR* $\kappa_d = \rho_{d\text{-}1}$ *XOR* $\kappa_{d\text{-}1}$, $[R_{SP}] = \rho_d$ *XOR* κ_d *XOR* $\kappa_d = \rho_d$. Из этого следует, что состояние регистров будет восстановлено на момент передачи управления на текущую подпрограмму и никакие целевые данные в памяти не будут искажены вследствие добавления кода средства защиты. Это позволяет сделать вывод о том, что оригинальная и модифицированная программы будут алгоритмически неразличимы при выполнении модификации путем вставки в зарезервированные участки такого кода.

Отметим, что переходы по адресам, содержащимся в регистрах, также не могут быть устранены или заменены. Используемые не в качестве начального

элемента *JoP*-атаки, они не являются самодостаточными без предшествующего помещения атакующим значений в регистры и передачи управления на соответствующие инструкции в рамках *RoP*-атаки. Следовательно, повышение устойчивости программы к *RoP*-атакам автоматически повышает устойчивость к *JoP*-атакам. При этом первичное получение контроля на ГПУ в рамках *JoP*-атаки возможно согласно модели атаки.

Рассмотрим возможные атаки, направленные на преодоление предложенных мер по защите возвратных инструкций.

Повреждение стека с перезаписью адреса. Для успешной реализации требует чтения значения ρ_d *XOR* κ_d и переполнения стека в рамках одной подпрограммы с записью в стек значения ρ_{gadget} *XOR* κ_d, где ρ_{gadget} – адрес первого гаджета в *RoP*-цепочке. При этом между утечкой значения защищенного адреса и применением эксплойта должно выполниться формирование полезной нагрузки эксплойта (то есть массива данных, который будет некорректно обработан и приведет к повреждению адреса возврата). Согласно модели атак, данный вариант принят как неактуальный.

Угадывание значения κ_d. При использовании *RDRAND* или *RDTSC* при неизвестном предшествующем значении вероятность 2^{-32} (вследствие использования 32-битного случайного значения). При известном значении *RDTSC* для вызывающей подпрограммы можно пессимистично оценить как 2^{-4}.

Повреждение стека вызывающей подпрограммы с заменой ρ_d *XOR* κ_d *XOR* κ_{d+1} на ρ_{gadget} *XOR* κ_d *XOR* κ_{d+1}. Данный вид атаки эффективен, но требует, согласно модели атаки, знания κ_d для корректного возврата из текущей подпрограммы для последующего перехвата управления в вызывающей подпрограмме.

Атака с чтением начальной области стека. Атакующий использует примитив чтения в *n*-й подпрограмме для получения значений κ_{init} *XOR* κ_0 и пар значений: $<\kappa_d$ *XOR* κ_{d-1}, κ_d *XOR* $\kappa_{d+1}>$, где $0<d<m$, $m<n$. Полученная последовательность не позволяет сформировать эксплойт для *d*-й подпрограммы, так как атакующему требуется значение κ_i, которое невычислимо из полученных пар при использовании качественного источника случайных чисел.

Атака с чтением фрагментов полного стека. Использование примитива чтения в *n*-й подпрограмме для получения значений $к_{init}$ *XOR* $к_0$ и пар значений: $<к_d$ *XOR* $к_{d-1}$, $к_s$ *XOR* $к_{d+1}>$, где $0<d<n$, и значения $к_n$ атака будет возможна, если все перечисленные значения могут быть получены без повторного вызова подпрограмм с *d*-й по *n*-ю, так как в противном случае будет выполнено пересоздание ключей на указанном участке. При пересоздании ключей в распоряжении атакующего будет фактическая последовательность: $к_{init}$ *XOR* $к_0$ и пары значений $<к_d$ *XOR* $к_{d-1}$, $к_d$ *XOR* $к_{d+1}>$ для $0<d<k$, пары значений $<к'_d$ *XOR* $к'_{j-1}$, $к'_j$ *XOR* $к'_{j+1}>$ для $k<j<n$. Индекс *k* соответствует границе прочитанных фрагментов стека. Чтение стека производится атакующими фрагментами согласно модели атаки. Полученная последовательность не позволяет сформировать эксплойт для *d*-й подпрограммы, так как требует значения $к_d$, для вычисления которого необходимы значения $к_{d+1}$, … $к_n$.

При наличии уязвимой к переполнению буфера подпрограммы и примитива чтения в подпрограммах не в единой последовательности вызываемых подпрограмм атака перестает быть возможна (то есть нахождение в разных ветках дерева вызовов ГПУ).

3.3. Алгоритм синонимической замены элементов программы, содержащих опасные значения

Под синонимической заменой элементов программы, содержащей ОЗ, будем понимать такую замену, чтобы состояние используемых регистров и ячеек ОЗУ после исполнения оригинального и модифицированного участка не отличалось для любого исходного состояния используемых регистров и ячеек ОЗУ на начало рассматриваемого участка. Критерий использования сформулирован далее.

Рассмотрим виды составных частей инструкций, которые могут содержать ОЗ. Непосредственно байт, который интерпретируется как ОЗ, не может входить в состав префиксов (так как префикс не может совпадать с существующими опера-

циями) и кодов операций первого уровня (тогда это будет ОИ, защита которых рассмотрена ранее). Согласно [80, 41], ОЗ могут встречаться в следующих участках инструкций:

– аргумент данных операции, являющийся ссылочной информацией, определяется после коррекции блочно-ссылочной структуры в соответствии с главой 2;

– параметры операции, задающие режим ее выполнения, или операнды (*ModRM*-, *SIB*-компоненты инструкции);

– аргумент данных операции, не являющийся адресом;

– второй и третий байт многобайтного префикса (значения 0x*C*2 и 0x*C*3 являются допустимыми значениями, а 0x*CA* и 0x*CB* неприменимы, так как необходимый им префикс 0x48 не входит в область допустимых второго байта трехбайтного префикса и не может являться первым байтом ни двухбайтового, ни трехбайтового префикса);

– код операции второго или большего уровня, интерпретируемый как опасная операция.

Рассмотрим алгоритм синонимической замены инструкций, содержащих ОЗ, обеспечивающий их устранение (за исключением ОЗ в ссылочной информации), который приведен на Рисунке 3.2. Алгоритм устранения ОЗ в ссылочной информации приведен в подразделе 3.2.2.

ОЗ в составе БД, для которых может быть снят атрибут исполнимости, переводятся в непригодное для атакующего состояние в соответствии с п. 3.2.3.

При вхождении ОЗ в состав непосредственного операнда, содержащего абсолютный адрес (мнемоника *moffset*64 в описании системы команд), существует два варианта нахождения адресуемого РЯ:

– В составе образа программы. При данном варианте нахождения устранения ОЗ осуществляется путем добавления байтов выравнивания перед каждым адресуемым РЯ в соответствии с п. 3.2.2 и интерпретации как локальной ссылки. Сходимость алгоритма обеспечивается тем, что при добавлении смещений в ходе обработки имеющих опасный адрес РЯ, адреса ра-

нее обработанных РЯ не изменятся. Данная мера защиты не приводит к изменению алгоритма или снижению производительности, но приводит к увеличению размера образа защищенной программы.

– Вне образа программы. При таком варианте размещения устранение ОЗ не может быть осуществлено путем перемещения фрагментов образа защищаемой программы. Для устранения применяется замена инструкции с операндом в виде абсолютного адреса на синонимичную инструкцию, считывающую тот же адрес из неисполнимой секции данных. В этом случае применяется *RIP*-относительная адресация с контролем отсутствия ОЗ в относительном адресе и устранение их при необходимости в соответствии с порядком, описанным далее в этом подразделе.

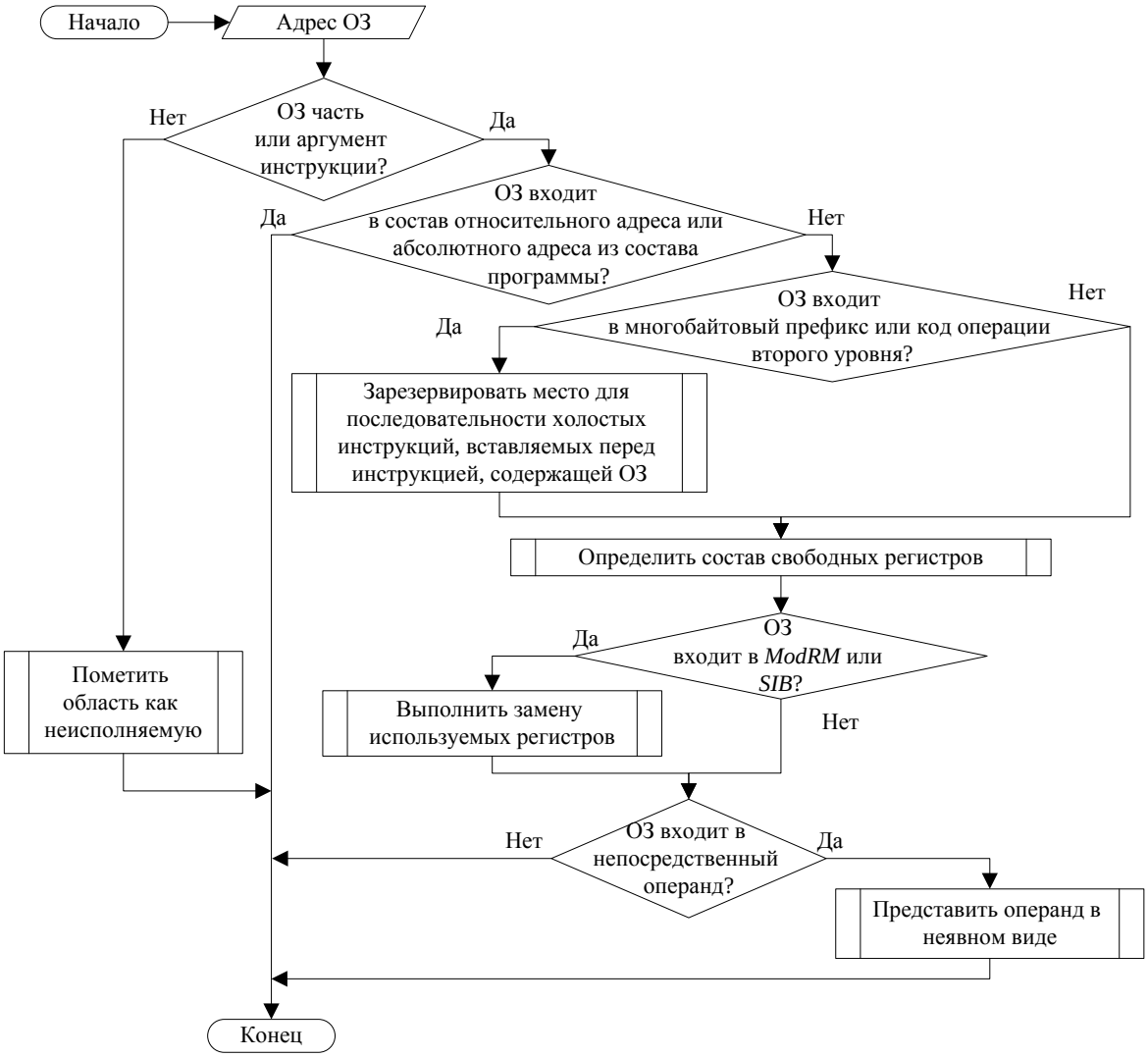

Рисунок 3.2. Алгоритм синонимической замены инструкций, содержащих ОЗ

Согласно системы команд *AMD*64, существует четыре таких инструкции (коды операций 0x*A*0–0x*A*3). Отметим, что в составе выборки проанализированных программ указанные инструкции, и, следовательно, такие способы адресации не применяются (статистика используемых инструкций приведена в приложении 3). Это соответствует формированию компилятором позиционно-независимого кода (*PIC*) и совместимости с технологией *ASLR*. Использование абсолютных адресов требует их коррекции при запуске, из-за чего данного решения избегают. Вследствие отсутствия реального использования в составе исполнимого кода абсолютной адресации для контроля корректности предлагаемых решений использовались синтетические тесты.

При нахождении ОЗ в префиксе или расширенном коде операции для его использования атакующим необходимо передать управление до начала ОИ. Поэтому для устранения данного гаджета достаточно перед инструкцией вставить цепочку однобайтовых инструкций, которые делают некорректное исполнение невозможным (например, *NOP*). Для определения размера необходимой вставки используется алгоритм *GALILEO*, для оценки возможности использования байта в инструкции, описанной ранее. Перебираются размеры от 1 до 14.

Когда ОЗ входит в состав аргумента данных инструкции или задает испозуемые её операнды , то она должна быть заменена на две и более инструкции. Их совокупное выполнение должно давать алгоритмически неразличимое преобразование состояний программы тому, которое осуществляла защищаемая инструкция. Результат замены не должен содержать ОЗ. В ходе такой замены необходимо следить за тем, чтобы выполнение результатов синонимической замены не оказывало влияние на логику целевой программы. Это требует использования только таких регистров или ячеек ОЗУ, которые не задействованы в целевой программе. Поиск свободных регистровых ресурсов, необходимый для всех иных случаев, описан в п. 3.2.1. Если таких нет, то требуется сохранение с последующим восстановлением ресурсов, для которых присутствует конфликт использования. Критерием отсутствия использования является то, что значение регистра в дальнейшем в рамках анализируемой подпрограммы будет утрачено вследствие затира-

ния другим значением без предшествующего этому чтения. Вследствие этого оно никак не может повлиять на работу программы. Место под замещающую последовательность выделяется тем же алгоритмом, что используется для резервирования места в ходе защиты эпилогов.

В рамках ОЗ в составе операндов алгоритмически неразличимая замена достигается путем замены операндов инструкции. ОЗ для байта *ModRM* состоит из: режима, равного 3 (операндами являются регистры), поля, определяющего регистровый аргумент *reg*, равного 2 или 3 (что соответствует регистрам *rdx/r*10 и *rbx/r*11), и поля *r/m*, равного 0 или 1 (что соответствует регистрам *rax/r*8 и *rcx/r*9). Для байта *SIB* аналогичные ОЗ должны содержаться в полях *base* и *index* с теми же значениями.

Для того чтобы сделать непригодным к использованию данный гаджет, достаточно заменить любой из операндов на регистр, который не входит в состав опасных. Если инструкция принимает только входной регистр или один из регистров является входным, а другой выходным, то неразличимой заменой будет вставка перед защищаемой инструкцией команды *MOV*, которая копирует данные из опасного регистра-операнда в свободный регистр, который не входит в состав опасных. В самой инструкции опасный регистр заменяется на свободный, который к моменту выполнения инструкции будет содержать то же значение, что и в оригинальной программе.

Если единственным операндом инструкции является выходной регистр или регистр, который одновременно является и входным, и выходным, то выполняется вставка двух дополнительных инструкций: перед – запись в свободный регистр значения из опасного, после – запись в опасный регистр значения из свободного. Отметим, что инструкции перемещения значений между регистрами не влияют на регистр флагов, что не нарушает логики работы программы. В самой инструкции производится аналогичная замена опасного регистра на один из свободных и не входящих в перечень опасных. Алгоритм определения свободных регистров приведен далее.

При устранении ОЗ в составе непосредственного операнда данных принципиальным является то, имеет ли использующая ОЗ в качестве операнда инструкция альтернативную форму, в которой вместо непосредственного значения используется операнд в регистре. Необходимо отметить, что подавляющее большинство инструкций, которые могут использовать непосредственный операнд, имеют альтернативную форму с регистровым операндом. Сведения о наличии альтернативных форм приведены в приложении 4.

Если альтернативная форма с регистровым операндом существует (например, инструкция *ADD*), то оригинальная инструкция заменяется на последовательность:

1. Если свободных регистров нет, то выполняется временное освобождение регистра путем записи его значения в стек (в конце последовательности значение должно быть восстановлено из стека).

2. Загрузка в свободный регистр преобразованного операнда. Для устранения ОИ операнд хранится в преобразованном виде (например, инвертированном или циклически сдвинутом) так, чтобы преобразованный вид не содержал ОЗ.

3. Восстановление значения операнда путем выполнения обратной операции – той, которая использовалась для его преобразования.

4. Выполнение регистровой формы оригинальной инструкции с оригинальным операндом.

Если не существует альтернативной формы оригинальной инструкции (например, инструкция *ENTER*), то производится дополнение со стороны меньших адресов защитными инструкциями, которые исключают неправильную трактовку защищаемой инструкции. Дополнительно необходимо учитывать особенности:

1. Не для всех инструкций ОЗ входит в область допустимых значений. Например, приведенная инструкция *ENTER* принимает в качестве аргументов два непосредственных операнда. ОЗ не может содержаться в младшем байте первого операнда, так как это нарушает выравнивание. Кроме того, ОЗ не может содержаться во втором операнде, так как его максимальное значение составляет 32. Маловероятно, что ОЗ будет содержаться в старшем байте первого операнда, так

как фреймы стека размером в 32 КБ представляют редкость. Вероятность появления снижает то, что ОЗ составляют 2 из 255 возможных значений старшего байта первого операнда, и то, что компиляторы в принципе не используют такие инструкции (что не сужает область применимости относительно указанной в третьем разделе).

2. Вторая и третья найденные инструкции, не имеющие альтернативной регистровой формы (*LWPINS* и *LWPVAL*), применимы только на этапе профилирования приложения и в генерируемом коде компилятора не встречаются (что не сужает область применимости относительно указанной в третьем разделе).

3. Иных инструкций, помимо перечисленных, только с формой, использующей операнд, согласно документации [80, 41], не существует.

Поиск существующих решений, позволяющих определить используемые программой ресурсы процессора, показал, что единственной публично доступной реализацией является [98], принцип описан в [99]. Анализ данного решения показал, что оно не учитывает БИП, вследствие чего каждый вызов подпрограммы приводит к пометке всех регистров как используемых, что порождает неверную оценку состава используемых регистров. Было принято решение разработать новый алгоритм определения свободных ресурсов, учитывающий БИП. Перед формулированием алгоритма рассмотрим теоретические сведения, необходимые для этого.

Для определения свободных ресурсов процессора необходим анализ полной последовательности инструкций подпрограммы, в которой находится подлежащая защите инструкция. Это связано с тем, что ряд регистров используется достаточно редко и при анализе неполного содержимого подпрограммы может быть сделан неверный вывод об отсутствии их использования. Анализу подлежат входные и выходные аргументы инструкций для определения регистров, которые могут быть задействованы в модифицированном участке и не повлияют на неразличимость результатов вычислений. Регистр может считаться свободным, если содержащееся в нем значение не может быть использовано в дальнейших вычислениях. Так

как описанные выше меры оперируют регистрами целиком, то для того, чтобы регистр считался свободным, должны быть свободны все разряды регистра.

Пример: значение, содержащееся в регистре после считывания, в дальнейшем не считывается повторно, а вместо этого перезаписывается другим значением. Приведенный пример показывает, что необходимо рассматривать свободные ресурсы строго с точки зрения шагов выполнения программы (то есть свободность ресурса может быть выявлена только для диапазона конкретных инструкций). Учитывая БИП, максимальным участком анализа разумно выбрать подпрограмму. Подпрограмма представляется в виде набора ББ, переходы между которыми осуществляются согласно ГПУ.

Определение свободности ресурсов в рамках ББ (то есть когда последовательность выполнения детерминирована) является тривиальной задачей. При необходимости определения в границах больших, чем ББ (например, при необходимости вставки перед первой инструкцией ББ), необходимо учитывать все возможные переходы на все рассматриваемые ББ (если достоверно не определено, что возможен только один вариант перехода).

Если в ходе анализа используемых ресурсов встречаются инструкции, создающие сторонние эффекты, то возможны следующие варианты действий:

– При анализе инструкций вызова подпрограмм из состава защищаемого приложения должен быть учтен БИП как максимально ограничивающий состав свободных ресурсов. БИП определяет, из каких регистров считываются данные, в какие регистры записывается результат исполнения, какие сохраняют свое значение и какие будут иметь неопределенное значение в результате вызова подпрограмм. При недостатке ресурсов может быть проведен дополнительный анализ использования регистров в начале и в конце вызываемой подпрограммы. Если соответствующие ресурсы находятся в свободном состоянии, то должен быть сделан вывод о том, что они свободны, до и после вызова анализируемой вызываемой подпрограммы.

– Если для системных вызовов и вызовов библиотечных функций не определен БИП, должно быть принято утверждение, что модифицируются все регистровые ресурсы.

– Если значения каких-либо ресурсов должны быть сохранены в интересах системы защиты, то до исполнения они должны быть сохранены (не обязательно непосредственно перед вызовом) и восстановлены на следующей инструкции, после создающей сторонние эффекты. Вследствие того, что в работе учитывается БИП, это дает априорную информацию об используемых ресурсах. Это позволяет сохранять только необходимые ресурсы, что снижает накладные расходы. Для ряда ситуаций сохранение значений непосредственно перед исполнением является недопустимым, так как нарушает неразличимость состояний. Например, рассмотрим сохранение единичного восьмибайтового значения непосредственно перед инструкцией *CALL*, которая передает управление на подпрограмму со стандартным прологом, которая принимает аргументы через стек. Тогда адрес *RBP*-18_{16} будет указывать вместо первого аргумента на сохраненное значение, что нарушит неразличимость алгоритмов программ. Такая проблема ассоциирована со стеком, так как код программ содержит массово работу по относительным адресам при передаче аргументов и хранении локальных переменных. Безопасны для помещения значений в стек только участки подпрограмм двух видов: а) после выделения всех локальных переменных и до выделения первого блока аргументов для вызова вложенной подпрограммы, а также б) после восстановления стека, следующего за вызовом подпрограммы и помещением в стек аргументов для следующей подпрограммы.

– Если модифицированный участок требует специфического набора ресурсов и они на участке вставки не являются свободными, то они должны быть сформированы путем сохранения текущих целевых значений в прологе модифицированного участка и восстановления в эпилоге. Примером является инструкция *RDTSC*, которая записывает результат своей работы в фиксированные регистры, что приводит к безвозвратной потере содержа-

щихся в них целевых значений. Теоретически, возможным вариантом являлось бы сохранение значений всех регистров (например, последовательностью операций *PUSH* или инструкцией *PUSHAD* для 32-разрядного режима), но это существенно повышает накладные расходы. А с учетом массовой вставки кода для защиты инструкция возврата замедляет программу гарантированно неприемлемым образом.

Необходимо отметить, что в анализе ресурса памяти в предлагаемой системе защиты нет необходимости, вследствие чего данный вопрос не рассматривался.

3.3.1. Алгоритм определения свободных ресурсов подпрограмм

Непосредственное определение свободных ресурсов, во-первых, сводится к определению регистров, запись в которые не приведет к нарушению неразличимость состояний программы, а во-вторых, записанные в интересах системы защиты значения не будут повреждены в ходе вычислений целевой программы. На основе данных утверждений можно сформулировать критерий отнесения ресурсов к свободным:

– регистр считается свободным до инструкции, которая произведет запись в него;

– если после записи значение из регистра не считывается до выполнения повторной записи, то регистр должен рассматриваться как свободный с момента первой записи. При этом самая первая операция должна считаться избыточной в части модификации регистра;

– после помещения в регистр неопределенного значения (например, после возврата из подпрограммы в несохраняемых регистрах, согласно БИП) он считается свободным до записи в него значения.

Модификация свободных регистров не влияет на выполнение программы. Вследствие этого их использование в рамках кода системы защиты гарантированно не модифицирует алгоритм оригинальной программы.

Состояние одного регистра не влияет на состояние другого. То есть используемые регистры определяются машинным кодом и не связаны с результатами

вычислений, так как архитектура команд не предусматривает непрямое указание регистров-операндов. Вследствие этого поиск свободных ресурсов допустимо проводить для каждого регистра независимо. Дальнейшее описание приведено для одиночного регистра, для получения полной информации процедура повторяется для всех необходимых регистров.

Пусть анализируемая подпрограмма состоит из набора инструкций f_j, $1 \le j \le n$, где n – количество инструкций в подпрограмме. Перед определением состояния ресурсов выполним подготовительные операции. Согласно документации на процессор и БИП, определим для каждой f_j, является ли анализируемый регистр входным аргументом (читается), и запишем результат в массив *RI* размером n, индексы элементов которого соответствуют индексам инструкций. Выполним аналогичное определение с точки зрения того, является ли регистр выходным, и запишем результат в массив *RO*. Элементы массивов могут принимать значения: используется (*Y*), не используется (*N*), не определен (*U*). Результат определения записывается в массив *RF*, имеющий аналогичный размер, элементы которого принимают значения: свободен (*F*), занят (*O*).

БИП определяет порядок передачи аргументов, но не все используемые для этого регистры могут быть задействованы. Для их определения может быть либо проведен анализ вызываемой подпрограммы, либо проанализирована семантика инструкций подпрограммы. При втором подходе все инструкции между записывающей в регистр неопределенное значение и потенциально считывающей его в качестве аргумента вызываемой подпрограммы должны быть помечены как не использующие регистр. Для этого вводится массив *RR* размером n, элементы которого принимают значения: инструкция предшествует чтению (*B*) или нет (*A*).

Для проведения анализа необходимы сведения о ГПУ подпрограммы. Множество инструкций разбивается на ББ $b_k = (j_{ek}, j_{bk})$, где j_{bk} и j_{ek} – индексы первой и последней инструкции ББ, $1 \le k \le n_b$, n_b – число ББ. Для ББ определяются связи между ними (для каждого определяется перечень ссылающихся на него ББ). Также определяется перечень эпилогов, которыми заканчивается выполнение анализируемой подпрограммы, $E = \{e_1, \dots, e_m\}$, где e – номера ББ, включающие эпило-

ги, а m – число эпилогов в подпрограмме. Для учета порядка обработанных блоков используется массив *PBB* размером n_b, элементы которого принимают значения: обработан (*P*), запланирован к обработке (*S*), не обработан (*U*, значение по умолчанию).

Разработанный алгоритм определения свободных ресурсов подпрограмм с учетом БИП приведен на Рисунке 3.3.

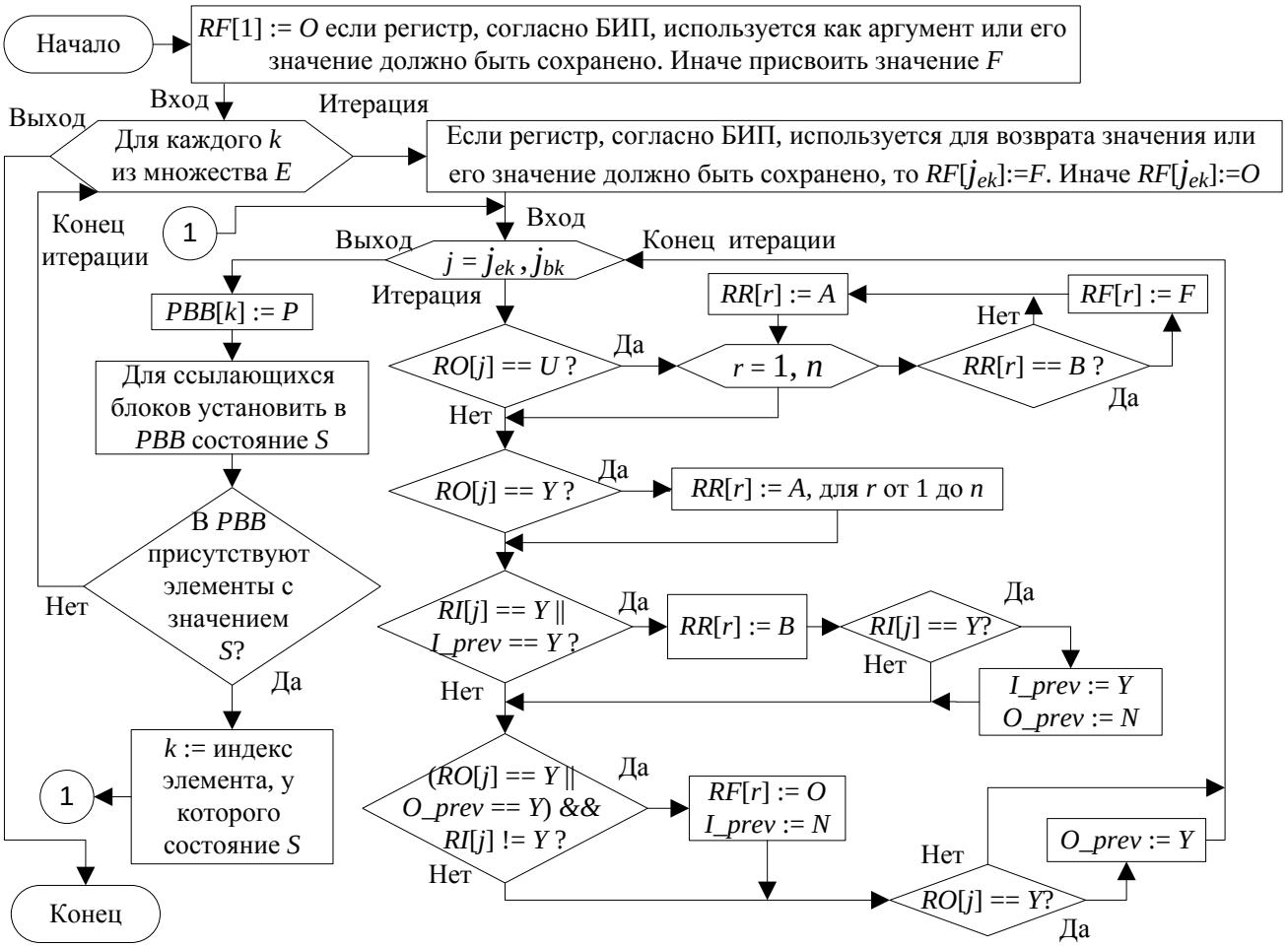

Рисунок 3.3. Алгоритм определения свободных ресурсов подпрограмм

В результате выполнения для каждого регистра указанного алгоритма каждой инструкции анализируемой подпрограммы будет сопоставлен перечень свободных ресурсов. Для ОИ он используется для формирования алгоритмически неразличимой замены.

При описанном выше порядке вызываемые подпрограммы рассматриваются как «черный ящик»; выдвигаются предположения о том, что для передачи аргу-

ментов и возврата значений используется максимально возможный состав регистров в рамках БИП. Состав входных и выходных регистров для вызова подпрограммы может быть уточнен при необходимости (например, при недостатке свободных ресурсов) путем анализа содержимого вызываемой подпрограммы. Порядок такого анализа совпадает с описанным ранее, а сведения о считываемых и модифицируемых ресурсах вызываемой подпрограммы интерпретируются как эффект от выполнения инструкции вызова подпрограммы.

Отсутствие же использования ресурса в рамках всех инструкций программы свидетельствует о том, что любые операции с ним не повлияют на алгоритмическую неразличимость программ. Если имеется априорная информация об отсутствии использования ресурса в рамках кода программы и не входящих в состав программы библиотек (например, вследствие ограничений на состав используемых компилятором инструкций) или гарантий отсутствия модификации, используемых системой защиты ресурсов системными библиотеками и системными вызовами, то она должна быть учтена для снижения времени анализа. Априорно не используемые ресурсы помечаются свободными.

3.3.2. Алгоритм устранения опасных значений в ссылочной информации

При вхождении ОЗ в состав относительного адреса необходимо учитывать семантику адресуемого РЯ и использующую его операцию для построения алгоритмически неразличимой их замены. Пересечением ссылок будем называть ситуацию, когда один и тот же байт для двух разных ссылок находится между базой и целевым адресом. Возможны следующие варианты, образующие полную группу, когда адресуются:

– БД (находящийся в другой, неисполнимой секции) или ББ без передачи управления (в рамках применения процедурных типов, например, для регистрации функций обратных вызовов или лямбда-функций). Наблюдаются множественные пересечения ссылок (например, все ссылающиеся ссылки на секцию данных будут пересекаться как минимум в старших байтах секции кода из-за последовательного их размещения).

– начало подпрограммы, на которую передается управление. Также будут наблюдаться множественные пересечения (например, таблица переходов к глобальным объектам размещается в начале секции кода, что приводит к пересечению всех ББ, передающих на нее управление как минимум в районе старших адресов этой таблицы);

– ББ в рамках той же подпрограммы. Пересечения или единичные, или отсутствуют вследствие того, что типовым является линейное построение подпрограмм.

Можно сделать вывод, что количество пересечений ссылок может быть оценено по критерию нахождения целевого и базового адреса в рамках одной подпрограммы (локальная ссылка). Устранение ОЗ для локальных ссылок достигается вставкой холостых инструкций (*NOP*, 0x90) между инструкциями или байт-заполнителями (*INT3*, 0x*CC*) в области выравнивания. Вставка выполняется по отрицательному смещению (если относительный адрес является отрицательным числом) или по положительному (если он положительный). Размер вставки выбирается таким, чтобы адрес после коррекции не содержал ОЗ. Процесс коррекции локальных относительных адресов проводится итерационно, так как вставка может привести к порождению нового ОЗ. Сходимость процесса обеспечивается тем, что количество подлежащих перемещению участков монотонно убывает из-за того, что, во-первых, число пересекающихся инструкций мало, а во-вторых, вероятность появления ОЗ равна 1/64 при равномерном распределении значений.

Для нелокальных ссылок описанный порядок не применим в общем случае. Рассмотрим оригинальную программу существенного размера, в которой корректируется некоторая ссылка, содержащая ОЗ. Пусть для каждого байта между базовым и целевым адресом имеется пересечение со ссылками, имеющими в составе младшего байта весь спектр возможных значений (от 0 до 255). Тогда коррекция на любое значение будет порождать столько новых ОЗ, сколько имеется соответствующих пересекающихся ссылок. При этом коррекция следующих ОЗ может привести к повторному появлению ОЗ в скорректированных ранее ссылках. Вследствие этого применение алгоритма корректировки локальных ссылок пере-

усложняет задачу. Алгоритм коррекции нелокальных ссылок, не содержащий описанного изъяна, не должен приводить к вставке каких-либо значений в секцию кода.

Предлагается решение, в рамках которого после секции кода помещается массив пятибайтовых записей (транзитная таблица). Структура записи: первый байт имеет значение 0xEB, соответствующее безусловному переходу по относительному адресу, а оставшиеся четыре байта содержат относительный адрес. Количество записей выбирается не менее, чем число нелокальных ссылок, содержащих ОЗ после коррекции всех локальных ссылок, и не менее, чем 0x200/5 байт, чтобы обеспечить возможность устранения ОЗ, содержащихся в битах адреса с индексами с 8 по 15.

Рассмотрим устранение ОЗ для нелокальной ссылки. Пусть база относительного адреса a_{base}, целевой адрес a_{target}. Выполняется следующий алгоритм:

1. Производится поиск первой неиспользуемой записи транзитной таблицы (все пять байт содержат нули).

2. Определить адрес записи a_{tt} и базовый адрес записи $a_{tt\_base} = a_{tt} + 5$. Значение «5» соответствует адресу следующей за текущей инструкции.

3. Рассчитать и проверить на наличие ОЗ значения $a_{tt} - a_{base}$ и $a_{target} - a_{tt\_base}$. При наличии в любом из значений ОЗ перейти к шагу 1, но начать поиск со следующей после текущей записи.

4. Заменить относительный адрес в защищаемой инструкции на первое рассчитанное значение, а в поле адреса из состава записи транзитной таблицы указать второе рассчитанное значение.

Приведенный алгоритм обеспечивает устранение всех ОЗ для нелокальных ссылок. При достаточном размере таблицы гарантируется нахождение такой записи, для которой оба рассчитываемых значения не будут содержать ОЗ без порождения новых ОЗ.

3.3.3. Исключение данных из состава исполнимой памяти

Для реализации этой меры защиты неисполняемые данные должны быть исключены из перечня исполнимых областей. Тогда содержащиеся в них байты с

ОЗ не могут быть использованы атакующими как гаджеты. Для определения атрибута исполнимости применяется технология *NX-bit*. Она позволяет разрешать исполнение инструкций только для конкретных участков программ (как минимум участка, содержащего машинный код). Перечень исполнимых участков содержится в заголовке образа программы.

Единицей указания атрибута исполнимости является страница памяти (в типовом случае размером 4096 байт). Для того чтобы в исполнимую память не были включены данные, применяется выравнивание. Перед участком кода и после него помещается массив байтов со значениями, которые не могут быть применены в составе гаджетов. Размер массива выбирается так, чтобы адреса начала и конца исполнимого участка памяти были кратны размеру страницы. В качестве значений байтов используется «0x*CC*», так как передача на них управления вызовет отладочное прерывание. Исходная секция, включающая данные до исполнимого кода, сам исполнимый код и данные после него, разбивается на три соответствующие секции, причем исполнимой (бит *NX* сброшен) является только вторая.

3.4. Выводы

В рамках главы приведен алгоритм устранения ОИ и ОЗ на участки, не пригодные для использования в качестве гаджетов. Для этого примняются вставки кода средства защиты, контролирующие целостность ГПУ, семантически алгоритмически неразличимые замены инструкций и исключение участков памяти из перечня исполнимых.

На основе результатов анализа гаджетов выделены участки программ, которые могут содержать ОИ и ОЗ. Используемая для защиты ОИ методика контроля целостности ГПУ основана на побитовом *XOR* адреса возврата с генерируемых при запуске каждого эпилога псевдослучайных значений. Раскрытие атакующим одного такого значения не позволяет проводить атаку для другой подпрограммы. Для повышения устойчивочти в наличии у атакующего примитива чтения используемые для контроля целостности значения хранятся в защищенном виде.

Для защиты от гаджетов, которые не содержат ОИ, предложен алгоритм синонимической замены элементов программы, содержащих ОЗ. Такие участки, не содержащие ссылочной информации, подвергаются заменене на аналогичные, но не пригодные для использования в составе гаджетов. Семантические замены предложены для всех инструкций, которые встречаются в программах. При выполнении коррекций ссылочной информации в соответстсвии с решениями, изложеными в главе 2, выполняется замена адресов, содержащих ОЗ. Для этого используются вставка холостых инструкций, изменение размеров участков выравнивания и транзитная таблица.

Для обеспечения корректной работы алгоритмов защиты ОИ и ОЗ разработан алгоритм определения свободных ресурсов процессора с учетом бинарного интерфейса приложений. Он позволяет выполнять обнаружение регистров, задействование которых для нужд системы защиты не влияет на алгоритм защищаемого приложения.

Таким образом, вставка кода системы защиты в соответствии с данной главой, обеспечиваяемая алгоритмом резервирования мест для встраивания кода средств защиты, образует первое защищаемое положение. Создано новое средство защиты, основанное на методе снижения числа пригодных для проведения *RoP*-атак участков в программах, обеспечивающее для программ в условиях отсутствия их исходных текстов снижение числа уникальных гаджетов на 98–100 %, а гаджетов, пригодных для атак – на 100 %. Порядок определения данных показателей приведен в следующей главе.

Методика контроля целостности графа потока управления обеспечивает защищенность при проведении атак через открытые компьютерные сети в условиях наличия у атакующего возможности взаимодействия с уязвимым приложением. Это составляет второе защищаемое приложение, заключающееся в обеспечении защищенности даже в условиях наличия у злоумышленника всей информации о программе, кроме используемых для защиты адресов возврата псевдослучайных значений.

4. Оценка эффективности средств защиты программ от *RoP*-атак

4.1. Существующие методы оценки эффективности средств защиты программ от *RoP*-атак

Методы оценки защищенности следуют из принципа построения систем защит. Рассмотрим данную специфику и соответствующие ей существующие методы:

– Рандомизация размещения (периодически [53], при старте [50], в родительских процессах [54]). Для данного подхода безопасность основана на неизвестности атакующему сведений о содержимом исполнимой памяти атакуемой программы. При наличии примитива чтения у атакующего программа становится уязвима перед атаками от момента получения достаточной информации и до следующего переразмещения, если таковое предусмотрено системой защиты. Метрики защищенности могут быть выражены как оценка вероятности наличия у атакующего примитива чтения и определение статистических показателей временно́го «окна», когда возможна атака. Такие метрики неприменимы для предлагаемой системы защиты, так как она построена на предположении известности атакующему информации о содержимом исполнимой памяти.

– Снижение числа пригодных для атаки гаджетов (на этапе компиляции [55, 56] в процессе работы за счет оверлеев [52]). Предлагаемое средство защиты является подвидом данного метода, и применяемые методы оценки рассмотрены далее.

– Затруднение получения контроля над ГПУ (путем контроля целостности адреса возврата [43], сигнатуры в стеке [100], косвенного указания адресов переходов [48]). Предлагаемое средство защиты частично использует данный метод для защиты ОИ, поэтому применяемые для оценки эффективности методы рассмотрены далее.

Проанализируем методы оценки эффективности систем защиты, используемые для двух последних принципов, для определения возможности применения в данной работе:

– Качественный анализ подверженности атакам. Основан на экспертной оценке путем соотнесения предлагаемых мер и модели атаки. Его недостатком является возможная неполнота рассмотрения, что может приводить к последующему выявлению недостатков защиты, пропущенных при авторском анализе [45] (возможным улучшением ситуации могло бы стать проведение аудита безопасности сторонней организацией, но такая практика не встречается в проанализированном материале).

– Подсчет числа гаджетов. Арифметический подсчет является корректным, когда ставится целью устранение всех ОЗ и ОИ. Такой подход применяется в [55, 56], хотя и без приведения детальной статистики неустраненных гаджетов. При защите ОИ вместо их устранения с целью сделать их непригодными для атаки такой подход не может показать корректного результата без доработки с целью учета не только количества, но и качества гаджетов (то есть возможности с их использованием провести атаку).

– Применение существующих эксплойтов (с адаптацией или без). Не является показательным в общем случае при изменении конфигурации фреймов стека (как, например, в предлагаемом принципе защиты), так как применимость зависит от их структуры. При этом потенциально возможно создание иного эффективного эксплойта, применимого к защищенной программе.

Из приведенных существующих методов только подсчет числа гаджетов оперирует объективными показателями, не зависящими от разработчика или экспериментатора, а также известности уязвимых мест программы. Рассмотрим недостатки данного метода требующие устранения для возможности сравнивать различные средства защиты, предназначенные для устранения или снижения числа пригодных для проведения атак гаджетов:

– если средство защиты устранило все гаджеты кроме одного последнего, но полезного для атакующего (имеющего, например, вид «pop edi; retn»), то при перехвате ГПУ после переполнения буфера атака через системный вызов «system» будет гарантированно достигать успеха;

– если средство защиты устранило лишь половину гаджетов, но оставшиеся имеют вид «nop; retn», то ни перехват ГПУ, ни полное раскрытие информации о содержимом ОЗУ не позволят провести успешную атаку;

– если два средства защиты оставили неустраненными одинаковую или сравнимую долю гаджетов, то существующий метод оценки эффективности покажет их равноэффективность. Для уточнения эффективности, учитывая показанную в двух предыдущих пунктах неравноценность гаджетов, потребуется применить либо метод экспертных оценок, либо попытаться применить эксплойт. Пример использования метода экспертных оценок содержится [55], хотя в данном источнике они содержат изъян – приведенный состав остаточных гаджетов актуален только при малом размере приложения (подробнее см. в 5.4).

– если средство защиты не ставит целью своего применения не устранение всех гаджетов, но невозможность проведения атак, то существующий метод оценки покажет его низкую эффективность, даже если они не будут пригодны для целей злоумышленника.

Учитывая приведенные выше проблемы требуется модифицировать метод подсчета числа гаджетов так, чтобы он учитывал не количество, а качество гаджетов и был применим ко всем доступным средствам защиты, направленным на устранение или перевод в непригодное состояние гаджетов. Модифицированный метод оценки эффективности строится на основе модели атак, приведенной в п. 1.2, которая определяет конечное состояние, к которому стремится привести значения регистров, содержащих аргументы подпрограмм, и R_{IP}.

4.2. Метод оценки эффективности систем защиты программ от *RoP*-атак

Для объективной оценки защищенности программ при применении предлагаемого и подобных ему решений необходимо определить количество реально пригодных для атак гаджетов. Такой подход позволит обойтись без субъективных экспертных оценок и экстраполяции существующих уязвимостей на защищаемую программу. Качественная оценка является неполноценной, так как оперирует предположениями о действиях атакующего, а не объективными данными, показывающими возможность атаки.

Применимость гаджетов в рамках атак определяется целью злоумышленника. Например, если атакующему необходимо вызвать функцию, принимающую аргументы через регистры *xmm*, то при отсутствии инструкций работы с ними в составе гаджетов конечная цель не может быть достигнута. Были проанализированы существующие эксплойты для определения конечных состояний, достижение которых является критерием успешности атаки. В качестве них были определены следующие:

– вызов подпрограммы из системной библиотеки, спроецированной на адресное пространство атакуемой программы. В стек атакующим помещаются данные, в регистрах передаются аргументы, в финале управление передается на целевую системную функцию (например, установки атрибута исполнимости на необходимую атакующему страницу памяти). Порядок передачи аргументов определяется БИП;

– для уязвимостей кода в привилегированном режиме – отключение аппаратных систем защиты процессора путем изменения режима работы процессора вследствие записи в управляющие регистры;

– простой передачи управления недостаточно для атаки, так как это соответствовало бы ситуации, когда в программе содержится участок кода, выполняющий все необходимые для злоумышленника операции (а это означает наличие программной закладки или НДВ).

Из анализа целей атакующего следует, что ему необходимо конкретное состояние системы для успешности атаки. Рассмотрим в качестве примера ситуацию, когда ему необходимо передать один аргумент в подпрограмму с указанием некоторого числа. Если атакующий может влиять на ГПУ, но не может записать в регистр *rdi*, используемый для передачи первого аргумента, необходимое значение, то атака не может быть проведена успешно. Соответственно, разрабатываемые метрики должны учитывать это.

Предлагаемый метод оценки эффективности сводится к сравнению показателя защищенности с нормой защищенности. Если норма защищенности не выполняется, то делается вывод о возможности атаки. Иначе атака невозможна.

Разработанный метод оценки должен учитывать, что аргументы работы с памятью имеют различную разрядность. Вследствие этого атакующий может располагать, например, средствами взаимодействия в ОЗУ или передачи управления в диапазоне адресов от 0 до 2^{32}-1. Данный диапазон штатно не используется ОС, вследствие чего все такие средства становятся бесполезными. Отсутствие учета таких аспектов ведет к неверной оценке результатов применения защиты.

В качестве исходных данных для метрик целесообразно использовать перечни гаджетов, формируемые существующими средствами их поиска. Это позволяет работать с тем же списком, что и атакующий. Корректность инструментария для поиска обеспечивается распространенностью использования, вследствие чего их ошибки устраняются.

При этом использовать напрямую результаты работы средств поиска гаджетов некорректно, так как не все показываемые ими гаджеты применимы и полезны. Необходимо учитывать только те гаджеты, которые приближают атакующего к состоянию, соответствующему критерию успешной атаки. Найденные гаджеты подлежат фильтрации, а лишь затем анализу.

Для учета специфики предлагаемого метода защиты гаджеты должны быть проанализированы на наличие штатных инструкций, модифицирующих вершину стека, которая используется как адрес возврата, и известность атакующему аргумента этих инструкций. Это отражает принцип рассматриваемой методики защи-

ты, когда перед инструкцией возврата производится вставка инструкции, выполняющей над адресом возврата операцию *XOR*. При этом аргумент операции генерируется случайно в прологе каждой подпрограммы. Если в рамках гаджета выполняется неизвестная операция, то вместо следующего гаджета управление будет передано на неопределенный адрес, что с близкой к 100 % вероятностью вызовет нештатное прекращение работы защищенного приложения. В результате уязвимость удаленного кода не будет реализована.

При разработке метрик должен быть учтен вопрос возможности транслировать численные показатели в возможность проведения атаки. Например, если атакующий имеет в распоряжении набор гаджетов, который не позволяет модифицировать память или регистры, но позволяет передавать управление, то такой набор не обеспечивает достижения цели атакующего. Таким образом, гаджеты, которые не позволяют менять состояние регистров или памяти, исключаются из рассмотрения и не должны влиять на метрики.

Изложенные выше особенности позволяют сформировать метрики:

– Число уникальных гаджетов после исключения из рассмотрения неприменимых гаджетов (критерии описаны далее, так как они зависят от программного средства выявления гаджетов). Данная метрика показывает общее количество потенциальных гаджетов. Её расчет важен для определения относительного снижения числа гаджетов при сравнении оригинальной и защищенной программ, а также исключения ложноположительного вывода о повышении защищенности для приложений, сформированных средствами защиты, интегрированными в компилятор (такими, как *G-free*).

– Перечень регистров, значение которых может задавать атакующий напрямую $\rho_{defined}$ (например, в результате извлечения данных из стека). Может быть определен как аргумент-назначение таких команд, как *MOV, POP* и т. п. Для краткости аргументы группируются по принципу вложенности (например, если атакующий может определять значение части регистра *ch* и регистра *ecx*, то достаточно указать значение *ecx*). Данная метрика показывает возможности атакующего задавать произвольные значения в регистрах

программы. Без её расчета невозможно определение способности атакующего задавать значения регистров и параметры алгоритма, исполняемого в ходе атаки.

– Перечень регистров, значение которых может задавать атакующий косвенно $\rho_{transit}$, – определяются регистры, на значение которых атакующий может влиять через выполнение операций или транзитных передач (из этого множества исключаются регистры, значение которых атакующий может задавать напрямую). Необходимость введения данной метрики обусловлена тем, что предыдущая метрика не отражает полного множества регистров, доступных атакующему.

– Показатель доступных подпрограмм $N\pi$, рассчитываемый на основе двух предыдущих метрик. При наличии возможности задать значение первого аргумента показатель равен единице, первого и второго – он равен двойке и так далее. Первые шесть аргументов передаются через регистры (указаны в порядке возрастания номера аргумента): *rdi*, *rsi*, *rdx*, *rcx*, *r*8, *r*9. Данный показатель отражает разнообразие возможных конечных подпрограмм, которые может вызывать атакующий (указывается для полноценно вызываемых подпрограмм и для частично определяемых аргументов, когда атакующий не может полностью определять значение регистра). При этом вследствие последнего критерия успешной атаки конечным состоянием не может быть вызов без аргументов. Так как атакующий имеет возможность управлять ГПУ (и, следовательно, вызывать любую подпрограмму), то возможность атаки сводится к возможности указания аргументов необходимой подпрограммы. Данная метрика является показателем защищенности.

На основании рассчитанных значений метрик делается вывод о принципиальной возможности или невозможности проведения атаки. То есть если атакующий не имеет средств для достижения конечного состояния атаки, то атака удаленного исполнения кода не может быть проведена. Иначе атака потенциально может быть реализована. Средствами для достижения конечного состояния атаки являются гаджеты, позволяющие задать значение регистров.

Норма защищенности имеет следующие значения:

– ноль – проведение *RoP*-атак невозможно, защита обеспечена;

– положительное значение – у атакующего есть потенциальная возможность задавать значения регистров, используемых для передачи аргументов, и возможность передачи управления на произвольный адрес.

Для предлагаемого средства защиты рассчитанный показатель защищенности должен иметь нулевое значение, так как ОЗ устраняются, а перед выполнением ОИ контролируется целостность ГПУ. Только тогда делается вывод об эффективности предлагаемого решения.

4.3. Алгоритм расчета метрик защищенности программ от *RoP*-атак

Рассмотрим алгоритм расчета метрик, приведенный на Рисунке 4.1.

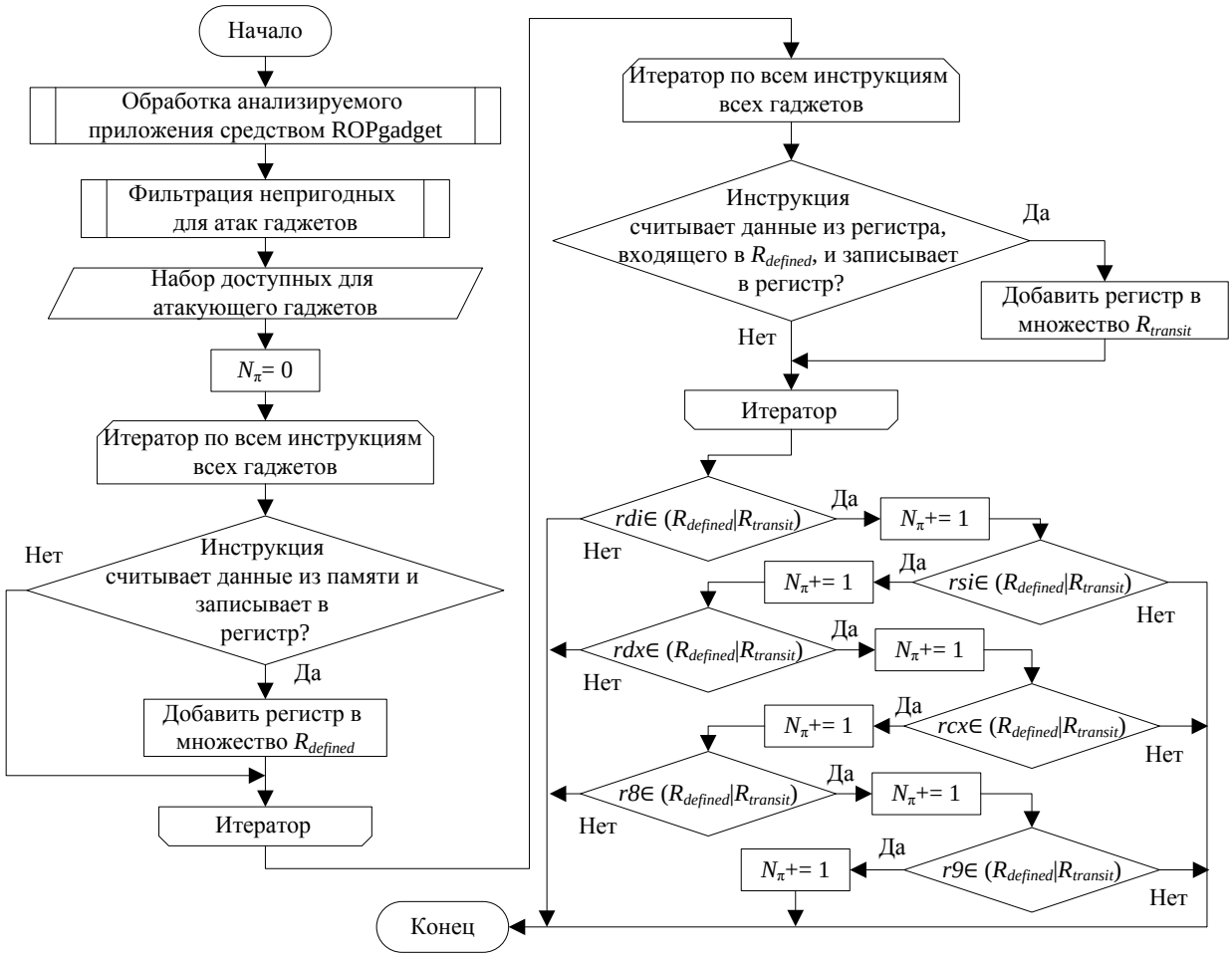

Рисунок 4.1. Алгоритм расчета метрик защищенности

Анализируемая программа обрабатывается как «черный ящик». Для получения первичных данных используется программное средство *ROPgadget* [101]. Его результаты работы записываются в файл. Для сформированных данных проводится фильтрация, в ходе которой устраняются ложно обнаруженные гаджеты. Удаляются из перечня при фильтрации гаджеты:

– Содержащие инструкцию *RETF*, если перед ней отсутствует *REX*-префикс «0x48», который задает 64-разрядный режим обработки. Это связано с тем, что при отсутствии префикса может быть задан только адрес в пределах 32-х бит с расширением нулями старших 32-х бит, но ОС не использует данный диапазон, что делает их ложно пригодными.

– Состоящие только из инструкций передачи управления, так как без модификации содержимого регистров и/или памяти атакующий не может достигнуть цели атаки. Возможна ситуация, когда без некоторой специфичной инструкции передачи управления построение *RoP*-цепочки станет невозможным. Определение таких ситуаций требует анализа конкретного сценария проведения атаки. Вследствие того, что он не может быть определен без рассмотрения конкретной уязвимости, то для отсутствия сужения применимости метрик будем считать, что у атакующего присутствуют в составе гаджетов все необходимые инструкции для передачи управления, а гаджеты, состоящие только из таких инструкций, исключаются из рассмотрения.

– Гаджеты, содержащие операции с неканоническими адресами (старшие два байта адреса должны быть равны 0 или 0x*FFFF*) или неиспользуемыми диапазонами адресов (первые 2^{32} байт).

– Выполняющие операции с неизвестным для атакующего и не модифицируемым им аргументом над адресом возврата. Например, в рамках предлагаемой системы защиты над адресом возврата проводится операция *XOR* с регистром *r*11, содержащим не известное атакующему значение. Если у злоумышленника нет возможности определить содержимое регистра, то исполнение такой инструкции в составе гаджета приведет к аварийному завершению программы.

Для отфильтрованных данных проводится анализ, в ходе которого формируются значения метрик, характеризующих защищенность от *RoP*-атак, и делается вывод о возможности или невозможности атаки.

4.4. Выводы

В главе приведено решение проблемы получения объективной оценки эффективности средств защиты от *RoP*-атак. Предложенный модифицированный метод оценки эффективности средств защиты программ от *RoP*-атак оценивает достижимость атакующим конечного состояния, при котором выполняется необходимая атакующему подпрограмма. Для этого определяется норма защищенности, отражающая тот факт, что при наличии у атакующего возможности вызывать подпрограммы хотя бы с одним аргументом позволяет говорить о факте полезного для атаки удаленного выполнения кода.

Приведенный в главе алгоритм расчета метрик защищенности программ от *RoP*-атак позволяет рассчитать показатель защищенности, отражающий количество аргументов подпрограмм, доступных атакующему. Из сравнения его с нормой защищенности делается вывод о возможностях атакующего вызова подпрограмм, и, как следствие, осуществимости *RoP*-атаки.

Описанные решения позволяют определять техническую реализуемость уязвимостей соответствующего типа, что составляет третье защищаемое положение.

5. Экспериментальная оценка эффективности разработанного средства защиты

5.1. Детали программной реализации

Декомпозиция решаемых комплексом задач обуславливает разбиение его на две независимые, абстрагированные друг от друга части: подсистему резервирования места для кода средства защиты (алгоритм работы описан в главе 2) и подсистему генерации синонимических замен (алгоритмы работы описаны в главе 3).

Подсистема резервирования места в оригинальной программе реализована в виде набора классов. Перечень адресов мест вставок и содержимое вставок обрабатываются без анализа. Общая схема взаимодействия компонентов, а также средство контроля качества приведены на Рисунке 5.1.

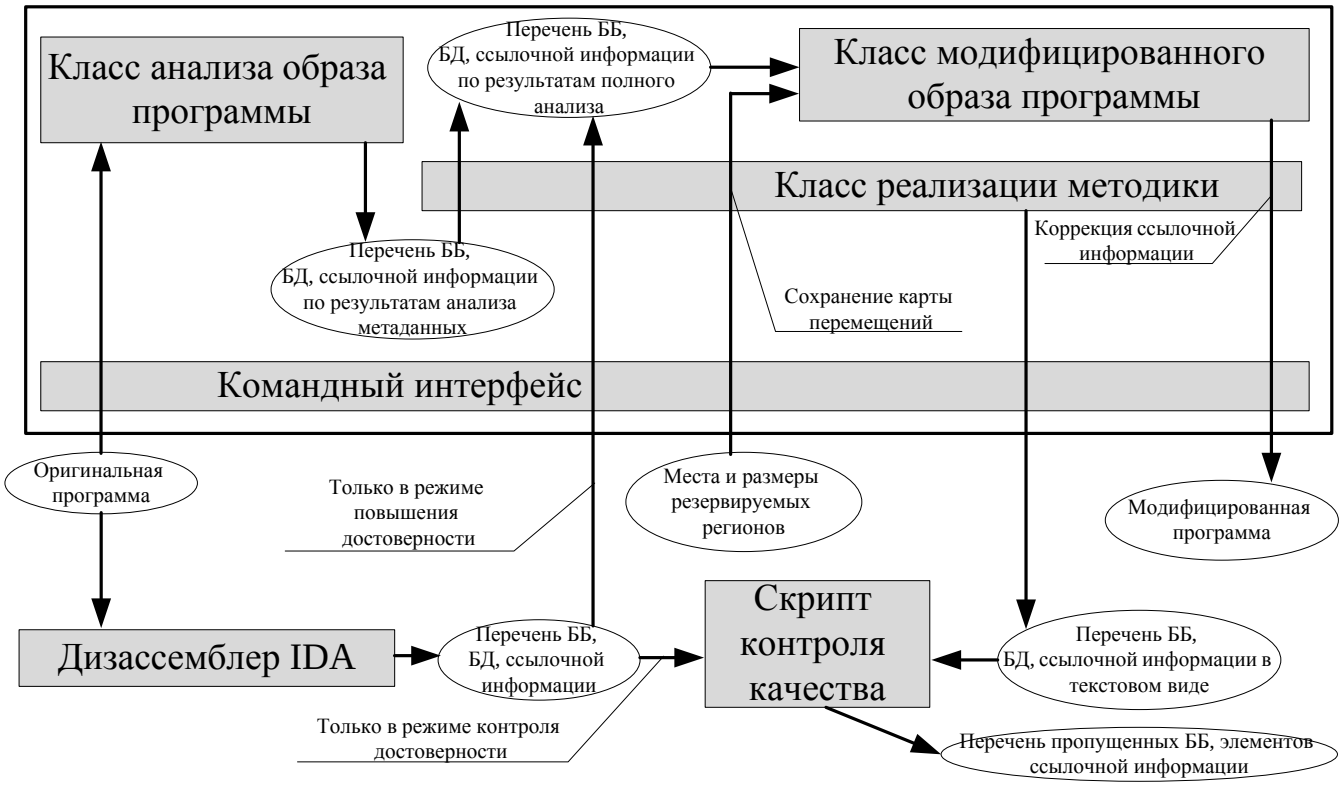

Рисунок 5.1. Схема взаимодействия компонентов подсистемы резервирования места

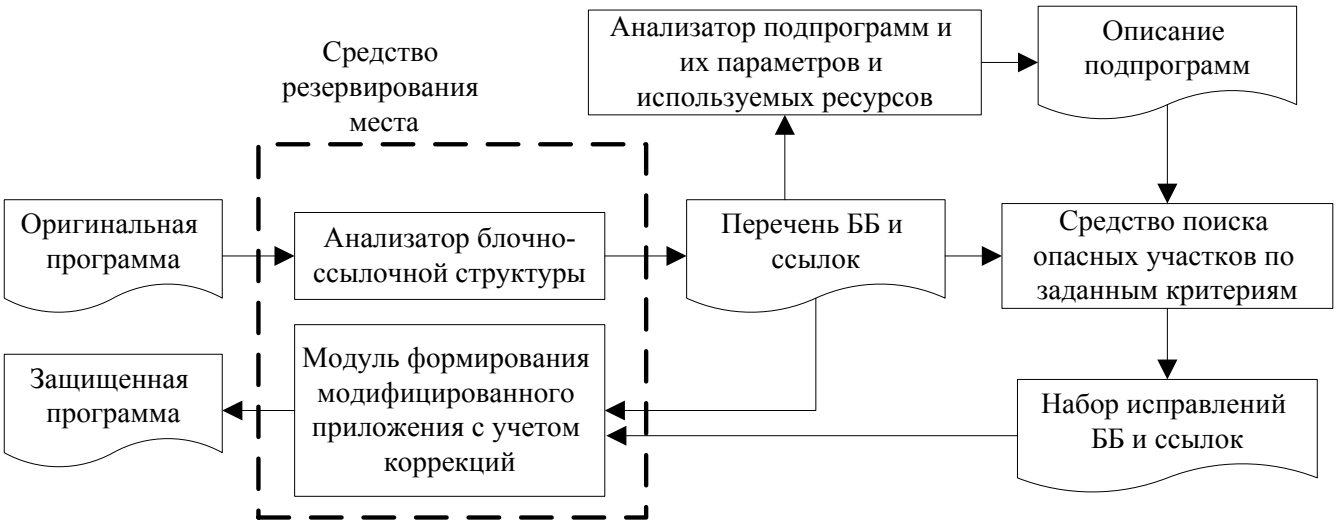

Рисунок 5.2. Подсистема генерации синонимических замен

Предлагаемые решения были апробированы на ОС *Debian* 9, относящейся к семейству *GNU*/*Linux*. Данная ОС использует БИП *ABI Linux-AMD*64 [38]. Для программной реализации использовалась среда *QtCreator*, компилятор *gcc* 6.3.0.

Для обеспечения автоматического тестирования ПС для управления был выбран интерфейс командной строки, а для вывода данных – текстовое представление. Посредством интерфейса командной строки задаются основные параметры, такие как имя оригинальной программы, имя выходного файла для записи защищенной программы, вид используемого источника $к_d$, детальность выводимой диагностической информации. Для отражения успешности применения системы защиты используется код завершения процесса (0 в случае успеха, иначе значение, указывающее на этап, вызвавший сбой). В консоль выводится диагностическая информация с настраиваемым уровнем детализации. Например, выводится информация о сегментах и секциях оригинальной программы, ББ, ссылках, найденных ОЗ, подпрограммах, а также об операциях по формированию защищенной программы. При сбое работы выводится диагностическая информация для устранения проблемы.

Для реализации защиты эпилогов используются вставки, представленные в Таблице 5.1. Приведен пример при использовании инструкции *RDRAND*. Защите подлежат только подпрограммы, хотя бы один эпилог которых заканчивается ин-

струкцией *RET* (не все подпрограммы заканчиваются такими инструкциями). Код приведенных вставок не содержит ОЗ.

Таблица 5.1. Вставки кода для защиты штатных инструкций возврата

Место вставки	Мнемоническое представление	Байтовое представление, шестнадц.	Примечание

В месте завершения пролога	rdrand r10
	movd r11, xmm14
	xor [r11], r10
	push r11
	lea r11, [rsp+**X**]
	movd xmm14, r11
	xor [r11], r10
	movd r11, xmm15
	xor r11, r10
	push r11
	movd xmm15, r10
		49 0F C7 F2
		66 4D 0F 7E F3
		4D 31 13
		41 53
		4C 8D 9C 24 **XX XX XX XX**
		66 4D 0F 6E F3
		4D 31 13
		66 4D 0F 7E FB
		4D 31 D3
		41 53
		66 4D 0F 6E FA
			X – смещение до адреса возврата; свободные РОН – *r*10 и *r*11; для хранения текущего ключа и адреса возврата выбраны не используемые в программе регистры расширения *xmm*15 и *xmm*14

Перед эпилогом	pop r9 pop r8 movd xmm14, r8	41 59 41 58 66 4D 0F 6E F0	Регистры *r*8, *r*9 свободны перед началом эпилога, согласно БИП

Перед инструкцией *RET*	movd r11, xmm15
	xor [r8], r11
	xor r9, r11
	movd xmm15, r9
	xor [rsp], r11
		66 4D 0F 7E FB
		4D 31 18
		4D 31 D9
		66 4D 0F 6E F9
		4C 31 1C 24
			Регистр *r*11 также свободен к завершению подпрограммы

В начале подпрограммы *_start*	rdrand eax
	push rax
	movd xmm14, rsp
	movd xmm15, rax
		0F C7 F0
		50
		66 4C 0F 6E F4
		66 4C 0F 6E F8
			Подпрограмма *_start* не требует защиты, так как заканчивается инструкцией *HLT*

Анализ вставляемого перед инструкцией *RET*-кода показывает следующие возможности интерпретации при передаче управления за указанное количество байтов до самой инструкции *RET*:

– 1 байт – последовательность *AND AL, C*3 (шестн. 24 *C*3), которая не может быть использована как гаджет (то есть байт *C*3 интерпретируется как часть инструкции *AND* и не может быть исполнен);

– 2 байта – *SBB AL,* 24_{16}; *RET* (шестн. 1*C* 24 *C*3) потенциально может быть использована как гаджет, выполняет вычитание из младшего байта регистра *RAX* константы 24_{16};

– 3 байта – *XOR* [*RSP*], *EBX*; *RET* – такая инструкция не может быть использована как гаджет, так как не модифицирует регистров, необходимых для вычислений или управления программой, но при известном атакующему значении в регистре *EBX* может быть использована для передачи управления на другой гаджет (путем записи на вершину стека значения R_{gadget} *XOR EBX* при предшествующем повреждении стека);

– более 3 байт – перед *RET* неизменно будет инструкция *XOR* [*RSP*], *R*11, которая преобразует значение адреса возврата на неизвестное атакующему значение (регистр *RSP* указывает на адрес возврата, так как защитный код вставлен перед инструкцией *RET*).

При применении описанного комплекса мер даже если атакующий получит возможность влиять на ГПУ (например, посредством атак на таблицы виртуальных функций), то в его распоряжении будет всего один гаджет, влияющий на РОН, не используемый для передачи аргументов в подпрограммы, и возможность передавать управление на другие такие же гаджеты, чего недостаточно для эксплуатации уязвимости, согласно модели атаки. Оценка защищенности для приложения в целом приведена далее.

5.2. Экспериментальная оценка корректности средства встраивания

Для синтетических тестовых программ получено корректное распознавание всех блоков, ссылок и участков выравнивания.

Сопоставление результатов работы программной реализации алгоритма и дизассемблера *IDA* версии 7.0.190307 показало:

– предлагаемая реализация обеспечивает более полное выявление ссылок для секций обработки исключений;

– дизассемблер *IDA* показал лучшее разделение участков данных и выравниваний в секции с данными только для чтения. Дальнейшее исследование показало, что это не является проблемой для применения алгоритма,

так как в соответствии с методикой перемещения смежные блоки данных перемещаются на одинаковое смещение, остаются смежными. Вследствие этого все ссылки, указывающие на участки рассматриваемой секции, будут скорректированы на одинаковое значение, соответствующее разнице адреса указанной секции до и после перемещения;

– в остальном результаты программных средств совпадают.

Для заимствованных из состава ОС программ ручная проверка модифицированных программ показала их работоспособность и отсутствие отличий от оригинальных. Для программ, снабженных модульными тестами, модификация не приводит к сбоям выполнения и не отражается на успешном прохождении тестов.

Результаты проверки согласно программе и методикам испытаний приведены в Таблице 5.2.

Таблица 5.2 – Результаты испытаний

№ п/п	Наименование проверки	Результат проверки
1	Проверка прохождения модульных тестов	Все тесты прошли успешно
2	Проверка обработки метаданных образов программ	Все секции обработаны корректно, блоки и ссылки из их состава выделены корректно
3	Проверка корректной обработки конструкции языка программирования *C* для уровня оптимизации 0	Все блоки и ссылки, содержащиеся в контрольном примере, обнаружены
4	Проверка корректной обработки конструкции языка программирования *C++* для уровня оптимизации 0	Все блоки и ссылки, содержащиеся в контрольном примере, обнаружены
5	Проверка корректной обработки конструкции языков программирования для уровней оптимизации от 1 до 3	Все блоки и ссылки, содержащиеся в контрольных примерах, обнаружены
6	Проверка работоспособности выборки программ из состава ОС после применения разработанного алгоритма	Не выявлено различий в работе программ из выборки
7	Проверка полноты анализа для выборки программ из состава ОС	Все блоки и ссылки, найденные *IDA*, обнаружены испытываемой программой
8	Проверка полноты коррекции для выборки снабженными тестами программ	Для проверок использовались *cppcheck*, *asn1c*. Проверки модифицированных программ прошли успешно
9	Проверка останова работы при выявлении нештатных ситуаций	Все сбойные ситуации были выявлены, анализ остановлен, а оператор – уведомлен

Пример применения методики для тестовой программы с комментариями приведен в Таблице 5.3. Ассемблерные листинги получены посредством дизассемблера *IDA*. Осуществлялась вставка 4096 байт по адресу 0x888. Числа слева от инструкций соответствуют адресу в шестнадцатеричной записи. Перечень обработанных приложений и результаты обработки приведены в Приложении 5.

Таблица 5.3 – Пример резервирования участков

Оригинальная программа	Модифицированная программа	Комментарий
Начальный участок подпрограммы main		
880 push rbp 881 mov rbp, rsp 884 sub rsp, 10h 888 mov [rbp+var_4], edi 88B mov [rbp+var_10], rsi	880 push rbp 881 mov rbp, rsp 884 sub rsp, 10h 888 nop … 1887 nop 1888 mov [rbp+var_4], edi 188B mov [rbp+var_10],rsi	Код до и после места вставки распознан дизассемблером одинаково
Начало технологической подпрограммы, расположенной по большим адресам, чем место вставки		
910 ; _libc_csu_init 910 push r15 912 push r14 914 mov r15d, edi 917 push r13 919 push r12	1910; _libc_csu_init 1910 push r15 1912 push r14 1914 mov r15d, edi 1917 push r13 1919 push r12	*IDA* корректно распознала имя подпрограммы, смещенное на 4096 байт
Начало технологической подпрограммы, расположенной по меньшему адресу, чем место вставки		
6E0 public _init_proc 6E0; CODE XREF: __libc_csu_init+2C↓p 6E0 sub rsp, 8	6E0 public _init_proc 6E0; CODE XREF: __libc_csu_init+2C↓p 6E0 sub rsp, 8	Перекрестная ссылка одинакова, что говорит о корректности преобразования

5.3. Экспериментальная оценка эффективности подсистемы синонимических замен

Предлагаемый метод защиты влияет на стек приложения, а для *RoP*-эксплойтов даже минимальное изменение расположения данных в стеке приводит к их неработоспособности. Вследствие этого проверка со сравнением работоспособности эксплойта для оригинального и защищенного приложения не является

показательной. Проверка работоспособности эксплойтов требует использования объективных метрик защищенности и приведена далее. Проверка корректности реализации проводилась по трем направлениям: неизменность логики работы программ, тестирование производительности и устранение пригодных для проведения атак гаджетов.

Для контроля неизменности логики работы применялись приложения (синтетические тесты, программы тестирования производительности и программы, снабженные модульными тестами), для которых исполняются все участки, для которых вносились изменения (это проверялось средствами контроля покрытия кода тестами *gcov*). Вследствие того, что выше приведено обоснование алгоритмической неразличимости вносимых изменений (то есть отсутствия влияния на логику работы), то для оценки корректности не важна вариативность входных данных. Неизменность логики работы оценивалась по идентичности выводимых программой данных при одинаковых входных (для оригинальной и защищенной программ) и отсутствию сбоев в работе. Результаты показали корректность и неизменность работы алгоритмов. Примеры алгоритмически неразличимых замен для устранения гаджетов приведены в Таблице 5.4. Перечень обработанных приложений и результаты обработки приведены в Приложении 5. Для контроля устранения ОИ и ОЗ использовалось средство их поиска *ROPgadget* [101]. Анализ показал, что во всех обработанных программах устранены оригинальные гаджеты.

Таблица 5.4. Примеры замен для устранения гаджетов

Содержимое памяти ориг. программы	Опасный участок ориг. прогр.	Опасный участок после эквив. замены	Содержимое памяти защищ. программы	Примечание
01 ***C*2**	ADD edx, eax	MOV rcx, rax ADD edx, ecx	48 89 *C*1 01 ***CA***	ОЗ в *ModRM*

48 83 ***C*3** 01	ADD rbx, 1	MOV rcx, rbx
		ADD rcx, 1
		MOV rbx, rcx
			48 89 *D*9
			48 83 ***C*1** 01
			48 89 *CB*
				ОЗ в *ModRM*

E8 **C3** FF FF FF	CALL -3D	NOP NOP CALL -3F	90 90 E8 **C1** FF FF FF	ОЗ в операнде *Imm*32

Прямое сравнение предлагаемого средства с аналогами не является в полной мере корректным, так как они рассчитаны на несовпадающую область применения и обеспечивают разную защищенность. Дополнительной проблемой является то, что для перечисленных в разделе 2 ближайших по ширине области применимости программных аналогов [53, 54] не опубликованы результаты их применения, вследствие чего возможно сравнение только на качественном уровне. К критериям сравнения (приведены в подразделе 1.3) добавлен «состав не анализируемых областей», так как он конкретизирует особенности средств, обеспечивающих устранение гаджетов. Результаты сравнения приведены в Таблице 5.5.

Таблица 5.5. Сравнение решений для противодействия *RoP*-уязвимостям

Средство защиты	Не анализируемые области	Устранение гаджетов	Контроль ГПУ	Уязвимость при наличии примитива чтения	Треб. исх. текст
PaXgrsecurity RAP	Исполнимые данные	Нет	Контроль с использованием одного неизвестного атакующему значения	Да	Да
G-free	Технолог. участки и исполн. данные	Да		Нет	Да
Обфускатор маш. кода	Технолог. участки и исполн. данные	В анализир. областях	Ковенная адресация	Да	Да
Shuffler	Ссылки после связывания и исполнимые данные	В части эпилогов		Между интервалами переразмещения	Нет
Selfrando	Отсутствуют	Нет	Нет	Да	Да
RuntimeASLR	Отсутствуют	Нет	Нет	Да	Нет
Scylla	Технолог. участки и исполн. данные	Нет	Нет	В рамках расшифрованной области	Да
Разработанное решение	Отсутствуют	Да	Уникальное для каждого запуска каждой подпрограммы значение	Использование примитива чтения и повреждения стека в рамках исполнения одной подпрограммы	Нет

Сравнение производительности выполнялось программным средством *coremark* [102]. Оно было выбрано, так как дополнительно контролирует целост-

ность результатов расчетов. Замеры производительности показали падение относительно оригинальной программы: при использовании *RDRAND* – 91 %, *RDTSC* – 53 %, ГПСЧ на отдельном ядре – 15 %, *AESENC* – 12,9 % (число запусков 100 для каждого). Характеристики тестовой системы: процессор *AMD A*6-6310 1,8 ГГц, 4 ядра, 4 ГБ ОЗУ, НЖМД 7200 об/мин.

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 (согласно [53]; реализация средства не доступна публично) составляют 15 %, худший результат для предлагаемого средства – 12,9 %. Остальные средства не устойчивы к атакам при наличии у атакующего примитива чтения. Для повышения достоверности сравнение дополнительно проводисось со средством *G-free* [55] с целью подтверждения применимости разработанных метрик защищенности к иным средствам защиты. Необходимо отметить, что данное средство не реализовано полноценно [103]. В частности, оно не позволяет корректно компилировать приложения для современных редакций языка Си++ и обрабатывать приложения при отсутствии их исходных текстов, чем обуслевлен неполный состав сравниваемых программ. При тестировании производительности описанным выше способом с применением *coremark* падение производительности составило 35 %.

5.4. Полученные метрики защищенности

Для проверки корректности методики оценки защищенности была сформирована выборка программ. Для каждой программы из выборки применяется обработка, реализующая предложенный методе снижения числа пригодных для проведения *RoP*-атак участков в программах. Для проверкки корректности методики оценки защищенности для оригинального приложения по возможности применяется средство защиты G-free. Для трех состояний каждой из полученных про-

грамм проводится определение метрик защиты. Полученные результаты сравниваются с показателем защищенности.

В состав выборки программ включены синтетические тесты и приложения из состава ОС для оценки защищенности до и после применения системы противодействия *RoP*-атакам. Примеры результатов определения метрик представлены в Таблице 5.6 (результаты средства защиты, реализующего предложенный в работе метод обозначены как «zpk»). Перечень обработанных приложений и полученные метрики защищенности приведены в Приложении 5. Из приведенных данных следует, что состав регистров, на которые может влиять атакующий, радикально уменьшается после применения предлагаемой системы защиты. Это приводит к тому, что атакующий хотя и может передать управление на целевую подпрограмму, но не имеет возможности указать ни один её аргумент. Из этого можно сделать вывод о том, что *RoP*-атака для удаленного исполнения кода нереализуема, а защита – обеспечена так как метрика защищенности $N\pi$ равна нулю. Для выборки в целом состав гаджетов после применения системы защиты соответствует приведенным примерам и не позволяет злоумышленнику проводить атаки.

Применение *G-free* не дало сопоставимого с предлагаемым средством результата, хотя и приводит к снижению числа гаджетов. Малое число приведенных программ связано с тем, что данная защита разработана для относительно старой версии компилятора, которая не позволяет выполнить сборку программ, написанных с использованием современных стандартов языков программирования (начиная с Си++11). Наблюдается тенденция: при возрастании размера защищаемого приложения увеличивается количество неустраненных гаджетов (что отражено в Приложении 5). При этом при наличии у атакующего примитива чтения доступный набор гаджетов позволит проводить успешные атаки.

Данный подход, в отличие от аналогов, основанных на экспертных оценках, оперирует объективными данными. Это устраняет человеческий фактор при анализе защищенности.

Таблица 5.6. Полученные метрики до и после применения защиты

Приложение	Уникальные гаджеты		Регистры, определяемые атакующим		Оперируемые регистры		Дост. арг. *N*π (полностью/частично)		Защита обеспечена
	Ориг.	zpk	Ориг.	zpk	Ориг.	zpk	Ориг.	zpk	
Синтетический тест, 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
Синтетический тест, 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

5.5. Описание внедрений результатов диссертационной работы

5.5.1. Применение разработанного средства защиты для программ на границе сетевого периметра АО «РТК-Сибирь»

В рамках своей коммерческой деятельности АО «РТК-Сибирь» необходимо обеспечивать удаленный доступ персонала для целей администрирования и передачи данных с удаленных рабочих мест. Для этого на стороне организации используется сервис *sshd* (входящего в *OpenSSH*) из состава AstraLinux 1.6 Special Edition. Данное приложение имеет примененную при компиляции защиту *StackGuarg*. Сервис имеет «белый» *IP*-адрес в сети Интернет, что позволяет проводить атаки на сервера организации и создавать угрозы её инфраструктуре. Вследствие невозможности применения средств защиты от узявимостей, требующих перекомпиляции, было применено разработанное средство защиты. Для проверки эффективности предложенного метода и его сравнения с уже используемой *StackGuard* было написано демонстрационное приложение. Это необходимо вследствие отсутствия открыто опубликованных эксплойтов для используемого приложения.

Демонстрационное приложение представляет собой сетевой сервер, принимающий подключения на *TCP*-порт 0x5555. В нем реализован примитивный сетевой протокол, принимающий три команды от клиента: чтения буфера, записи в

буфер и завершения сеанса работы. Код уязимого приложения приведен в Приложении 6. Последняя используется для штатного закрытия сокета. Остальные две рассмотрим подробнее.

Реализация команды чтения имеет заложенный дефект: клиент может указать размер считываемых данных, которые могут быть больше, чем буфер данных. Примером реальной уязвимости, которая дает аналогичные возможности атакующему, является *HeartBleed* в *openssl*. Это соответствует наличию у атакующего пимитива чтения, который позволяет:

– преодолеть *ASLR* (как для приложения, так и для системных библиотек – за счет содержащихся в стеке адресов возврата);

– раскрыть внутренние данные *StackGuard* (за счет раскрытия значения так называемой *stack canary*, которое служит для контроля целостности адреса возврата);

– раскрыть внутренние данные, формируемые предложенной системой защиты.

Реализация команды записи позволяет атакующему выполнить переполнение буфера в стеке за счет отсутствия контроля размера принимаемых данных. Примером является любая уязвимость, основанная на слабости ПО с идентификатором *CWE*-119 (отсутствие проверки границ при записи в буфер). Используя данную уязвимость, атакующий может переписать данные в стеке (в диапазоне 1024 байт).

Компиляция приложения осуществлялась с ключом «*-fstack-protector-all*», что обеспечивает добавление кода защиты *StackGuard* во все подпрограммы.

Далее был сформирован эксплойт, обеспечивающий в рамках первого отправленного пакета раскрытие содержимого стека (в том числе базового адреса загрузки образа уязвимомого приложения в ОЗУ), и в рамках второго пакета – перехват контроля над ГПУ и вызов подпрограммы *system* с одним аргументом. Таким образом, показано, что *StackGuard* не обеспечивает защиту при наличии в программе примитива чтения и возможности у атакующего перехватить ГПУ.

Таблица 5.6. Структура стека при проведении атаки

Смещ., шестн.	Семантика в программе	Содержимое, шестнадцатеричное		Примечание
		Оригинальное	Перезаписываемое	
+0*x*00	*val*[0], *val*[1]	Незначимо	Незначимо	
+0*x*08	*val*[2], *val*[3]	Незначимо	Незначимо	
+0*x*10	result, padding	Незначимо	Незначимо	
+0*x*18	stack canary	008459*b*7*b*6*d*6*e*92*b*	008459*b*7*b*6*d*6*e*92*b*	Меняется каждый запуск программы
+0*x*20	Значение *rbp* вызывающей подпрограммы	*c*09*e*06*e*9*ff*7*f*0000	незначимо	Определяет адрес базы стека
+0*x*28	ρ_d	load base + d6b	load base + e23	Гаджет pop rdi; ret
+0*x*30	Незначимо	Незначимо	<адрес команды>	Запишется в *rdi*
+0*x*38	Незначимо	Незначимо	<адрес подпрограммы *system*>	
+0*x*40	Незначимо	Незначимо	<команда атакующего, подаваемая в качестве аргумента>	

После защиты программы в соответстсвии с предложенной методикой средством защиты по смещениию 0*x*28 относительно начала буфера были добавлены два восьмибайтовых поля: защищенный адрес возврата вызывающей подпрограммы (ρ_{d-1} *XOR* κ_d) и защищенный ключ, сгенерированный в предыдущей подпрограмме (κ_{d-1} *XOR* κ_d). Также выполняется преобразование ρ_d в ρ_d *XOR* κ_d. Так как κ_d генерируется при запуске уязвимой подпрограммы и не содержится в явном виде в стеке, то у атакующего нет возможности построить эксплойт с использованием κ_d, кроме как угадав его. При этом каждая неудачная попытка приводит к аварийному завершению программы.

Также для оригинальной и защищенной программы были проанализированы метрики защищенности. Для оригинальной программы было найдено 165 гаджетов, применение которых позволяет атакующему запускать подпрограммы с произвольным числом аргументов. Для защищенной программы атакующий не может задавать ни одного аргумента, что говорит о невозможности проводить *RoP*-атаки.

После подтверждения эффективности предложенных мер разработанное средство защиты было применено для приложения *sshd*. Полученные метрики защищенности приведены в Приложении 5. Неразличимость защищенного сервер-

ного приложения от исходного приложения проверялась в ходе взаимодействия клиентской части приложения. Значимого снижения производительности пользователями не отмечено.

5.5.2. Внедрение системы защиты в процесс разработки ПО в СФУ

Защита от уязвимостей является одной из составляющих частей разработки современного ПО. При предъявлении заказчиком требований к данному направлению такие меры становятся обязательными к реализации для успешного выполнения контрактов. СФУ ведет разработку СПО для ОССН *Astra Linux Special Edition*. Входящие в её состав системные библиотеки и средства разработки используют защиту от повреждения адреса возврата подпрограмм вследствие переполнения буфера *StackGuard*. Данная система защиты уязвима, например, при переполнения буфера в куче с последующим повреждением указателя на таблицы виртуальных функций из состава полиморфных объектов языка Си++ или адресов функций обратного вызова в структурах языка Си.

Для защиты от переполнения буфера в куче в указанной ОССН используется сигнатура (которая по умолчанию не используется), размещаемая после выделяемого блока данных. Значение сигнатуры в рассматриваемой ОССН имеет размер в один байт и формируется примитивным ГПСЧ. При этом для генерации следующего значения используется предыдущее. Таким образом, атакующий может либо по раскрытому ранее значению однозначно определить значение между переполняемым блоком и целевым, либо максимум за 256 попыток угадать необходимое значение. При этом проверка сигнатуры выполняется либо при освобождении блока, либо путем вызова специальной подпрограммы *mcheck*. Если между проведением переполнения буфера и вызовом любой виртуальной функции из поврежденного объекта не будет выполнено освобождение повреженного блока, то сохранение сигнатуры не требуется. Столь слабая защита обусловлена тем, что она направлена на выявление непреднамеренного повреждения данных в куче. Таким образом, у атакующего есть возможность перехвата контроля над ГПУ при попытке использования поврежденных данных программой.

В ходе динамического анализа,включающего фаззинг и направленного на обнаружение уязвимостей в разрабатываемом ПО, из состава найденных были отобраны две ошибки переполнения буфера в куче. Обе были связаны с записью данных, пришедших по сети, без проверки их фактического размера в буфер, хранящийся в объекте с фиксированной емкостью *std::vector<uint*8_*t*>. Для выявления использовались следующие программные средства: в рамках модульных и интеграционных тестов – *valgrind*, а для фаззинга – *AFLplusplus*.

В случае обеих уязвимостей по большим адресам относительно переполняемых буферов располагались объекты, содержащие виртуальные функции. В Таблице 5.7 приведена структура области кучи, содержащая переполняемый буфер и размещенный после него объект с таблицей виртуальных функций.

Таблица 5.7. Структура области кучи при проведении атаки

Смещ., шестн.	Семантика в программе	Содержимое, шестнадцатеричное		Примечание
		Оригинальное	Перезаписываемое	
+0*x*00	std::vector<uint8_t> .data()	незначимо	Первые 56 байт – незначимо 8 байт – адрес первого гаджета в цепочке	Адрес должен быть раскрыт атакующим до атаки для преодоления *ASLR*
+0*x*400	Сигнатура	41	41	Подбирается в ходе атаки
+0*x*401	Служебные данные следующего блока	00 00 00 00 00 00 00 00 00 00 00 00 00 00 0*F* 21 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 0F 21 00 00 00 00 00 00 00	Всегда фиксировано в рассматриваемом случае
+0*x*420	ModuleInterface::_vptr	Адрес таблицы виртуальных функций класса *ModuleInterface*	Адрес буфера (соответствует смещению +0*x*00 в данном примере)	
+0*x*428	поля объекта с типом *ModuleInterface*	незначимо	не изменяется	
+0*x*534	Сигнатура	45	Не изменяется	

Без применения предложенных в диссертации мер защиты атакующий максимум за 256 попыток подбора сигнатуры (фактически понадобилось 27 попыток для первой уязвимости и 65 попыток для второй) перехватывает контроль над ГПУ. Дальнейшая атака аналогична описанной в п. 5.5.1 и связана с построением

цепочки гаджетов для вызова подпрограммы-цели атакующего с нужными аргументами.

После обработки защищаемого приложения и используемых им библиотек разработанным средством противодействия *RoP*-атакам было проверено, что перехват контроля над ГПУ также возможен (вследствие того, что целью является устранение гаджетов, используемых после перехвата). При этом за счет применения метода снижения числа пригодных для проведения *RoP*-атак участков в программах были устранены гаджеты, основанные на исполнении со сдвигом, позволяющие задать хотя бы первый аргумент подпрограммы. Примененная методика контроля целостности ГПУ обеспечила невозможность использовать код эпилогов программы и бибилотек для реализации атаки. Перечень гаджетов для оригинального приложения формировался средством *RoP-gadget*. Таким образом, предложенный метод защиты хоть и не устраняет сами уязвимости в программном обеспечении, но не позволяет атакующему провести успешную атаку. Снижение производительности защищенного приложения составило 3 %, что не является существенным с точки зрения его требований назначения.

5.5.3. Внедрение результатов работы в образовательный процесс СибГУ

Результаты диссертационной работы были внедрены в учебный процесс для студентов кафедры безопасности информационных технологий СибГУ им. М.Ф. Решетнева в рамках модуля «Уязвимости программного кода и защита от них» курсов «Безопасность операционных систем», «Аппаратные средства вычислительной техники» и «Организация ЭВМ и вычислительных систем». Программное средство позволяет на практике познакомить студентов с причинами возникновения и способами эксплуатации уязвимостей, а также современными подходами в области противодействия им и оценки защищенности.

Программное средство позволяет выполнять выделение низкоуровневых структурных элементов программ, выявлять уязвимые участки и формировать их алгоритмически неразличимые замены. Сравнение проведенных атак на оригинальные и защищенные программы демонстрирует студентам механизм эксплуатации уязвимостей с целью удаленного исполнения вредоносного кода.

В свою очередь, средства встраивания в программный код и средства расчета метрик защищенности исполняемого кода применяются студентами в рамках их собственных научно-исследовательских проектов, направленных на низкоуровневый анализ исполняемого кода, таких как тестирование ПО и создание систем защит.

Комплект практических заданий, разработанный на основе выполненной работы, включает (помимо методических материалов):

– пример уязвимого приложения, содержащего примитив чтения и ошибку переполнения в стеке. Данный образец позволяет изучить порядок проведения атак без применения мер защиты и демонстрирует опасность уязвимостей удаленного исполнения кода;

– пример уязвимого приложения, аналогичный предыдущему, но содержащий уязвимость переполнения буфера в куче;

– приложения, основанные на двух предыдущих пунктах, но защищенные средством *StackGuard*. Демонстрируют недостаточность стандартных мер защиты при наличии в программах критических уязвимостей;

– два приложения на основе предыдущих, но обрабатанные предлагаемым в работе средством защиты. Демонстрируют невозможность проведения *RoP*-атак при устранении гаджетов из программного обеспечения.

Кроме того, разработанные алгоритмы использовались при подготовке заданий для соревнований по практическим аспектам информационной безопасности в формате *Capture the Flag* для школьников и студентов, запланированных в 2022 году.

5.6. Выводы

В первом подразделе настоящей главы приводится описание подсистем и их взаимодействия в составе программного комплекса. Для повышения гибкости и масштабируемости вся система была построена по модульной архитектуре с абст-

рагированием частей друг от друга. Такой подход позволяет обеспечить модернизационную пригодность в рамках дальнейшего развития комплекса. При этом стоит отметить, что все компоненты системы (за исключением блока анализа образов приложений и блока выделения подпрограмм) могут использоваться как в *Windows*, так и в ОС семейства *Linux*.

В заключительном разделе приводится описание внедрения результатов работы в рамках защиты приложений сетевого периметра АО «РТК-Сибирь» и в процесс разработки ПО в ФГАОУ ВО «Сибирский федеральный университет». Кроме того, программный комплекс используется в учебном процессе на кафедре безопасности информационных технологий СибГУ им М.Ф. Решетнева.

ЗАКЛЮЧЕНИЕ

В рамках работы были модифицированы существующие методы защиты и методы оценки эффективности. На их основе разработаны и реализованы алгоритмы и средство защиты, позволяющие предотвратить успешное проведение *RoP*-атак для программ.

По результатам проведённой работы можно сделать следующие выводы:

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 возможности 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 определения свободных ресурсов процессора с учетом бинарного интерфейса приложений позволяет выполнять обнаружение регистров, задействование которых для нужд системы защиты не влияет на алгоритм защищаемого приложения.

4. Разработанный алгоритм синонимической замены элементов программы, содержащих опасные значения, которые могут быть использованы в составе гаджетов, и методика контроля целостности графа потока управления, обеспечивающая защиту штатных инструкций возврата из состава эпилогов подпрограмм, позволяют выполнять устранение опасных участков без искажения алгоритма защищаемого приложения.

5. Проведенная экспериментальная оценка результата применения предлагаемого метода защиты показала падение производительности на 13 %, что нахо-

дится на уровне ближайших аналогов, которые обеспечивают меньшую защищенность.

6. 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, что препятствует достижению злоумышленником целей.

7. 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.

Дальнейшие перспективы работы в рамках рассмотренной тематики заключаются в расширении области применения для приложения ОС *Windows*. Также перспективным направлением является замена ненадежных защит типа *StackGuard* на предлагаемое решение без перекомпиляции.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

РФ – Российская Федерация.

ПО – программное обеспечение.

НДВ – недекларируемые возможности.

ГПУ – граф потока управления.

ОС – операционная система.

ББ – базовый блок.

БД – блок данных.

CVE – The Common Vulnerabilities and Exposures.

СЛОВАРЬ ТЕРМИНОВ

Бинарный код, машинный код, исполняемый код: код, исполняемый процессором.

Дефект, ошибка: каждое отдельное несоответствие продукции установленным требованиям.

Тестирование: деятельность, направленная на обнаружение дефектов в программном обеспечении.

Полнота тестирования: отношение количества выполненных в ходе тестирования базовых блоков программы к общему количеству достижимых базовых блоков в программе.

Уязвимость информационной системы, брешь: свойство информационной системы, обусловливающее возможность реализации угроз безопасности обрабатываемой в ней информации.

Программное обеспечение: совокупность программ системы обработки информации и программных документов, необходимых для их эксплуатации.

Исходный код: компьютерная программа в текстовом виде на каком-либо языке программирования.

Уязвимость в ПО, уязвимость в программном коде: ошибка, обуславливающая возможность реализации угроз безопасности к обрабатываемой в ПО защищаемой информации.

Проприетарное (закрытое) программное обеспечение: программное обеспечение, распространяемое на условиях простой (неисключительной) или исключительной лицензии, ограничивающей использование программы и/или запрещающей пользователю внесение изменений в программу для ЭВМ (переработку) и/или распространение изменений (переработанной программы).

Надежность программного обеспечения: способность программного продукта безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью.

Несанкционированный доступ к информации: доступ к информации, нарушающий правила разграничения доступа с использованием штатных средств,

предоставляемых средствами вычислительной техники или автоматизированными системами. *Примечание. Под штатными средствами понимается совокупность программного, микропрограммного и технического обеспечения средств вычислительной техники или автоматизированных систем.*

Программная закладка: скрытно внесенный в программное обеспечение функциональный объект, который при определенных условиях способен обеспечить несанкционированное программное воздействие. Программная закладка может быть реализована в виде вредоносной программы или программного кода.

Недекларированные возможности: функциональные возможности ПО, не описанные или не соответствующие описанным в документации, при использовании которых возможно нарушение конфиденциальности, доступности или целостности обрабатываемой информации.

Конфиденциальность информации: обязательное для выполнения лицом, получившим доступ к определенной информации, требование не передавать такую информацию третьим лицам без согласия ее обладателя.

Целостность информации: состояние информации, при котором отсутствует любое ее изменение либо изменение осуществляется только преднамеренно субъектами, имеющими на него право.

Доступность информации: состояние информации, при котором субъекты, имеющие права доступа, могут реализовать их беспрепятственно.

Угроза: совокупность условий и факторов, создающих потенциальную или реально существующую опасность нарушения безопасности информации.

Модель угроз: физическое, математическое, описательное представление свойств или характеристик угроз безопасности информации.

Буфер: рабочая область памяти при пересылке данных.

Стек: динамическая структура данных, представляющая из себя упорядоченный набор элементов, в который добавление новых элементов и удаление существующих производится с одного конца, а извлечение – в обратном порядке их добавления в структуру.

Язык ассемблера: язык программирования, который представляет собой символьную форму машинного языка с рядом возможностей, характерных для языков высокого уровня.

Язык высокого уровня: язык программирования, понятия и структура которого удобны для восприятия человеком.

Компиляция: трансляция программы с языка высокого уровня в форму, близкую к программе, на машинном языке.

Компилятор: программа или техническое средство, выполняющее компиляцию.

Дизассемблирование: процесс и/или способ получения исходного текста программы на ассемблере из программы в бинарном коде.

Дизассемблер: транслятор, преобразующий машинный код в программу, на языке ассемблера.

Эксплойт: программа, фрагмент программы или последовательность команд, использующие уязвимость/уязвимости программного обеспечения для нарушения безопасности автоматизированной информационной системы.

СПИСОК ЛИТЕРАТУРЫ

1. БДУ ФСТЭК [Электронный ресурс]. – Режим доступа: https://bdu.fstec.ru/vul, свободный (дата обращения: 01.09.2021).

2. Exiting x86: Why Apple and Microsoft are embracing the Arm-based PC [Электронный ресурс]. – Режим доступа: https://siliconangle.com/2020/06/26/exiting-x86-apple-microsoft-embracing-arm-based-pc/, свободный (дата обращения: 01.09.2021).

3. The Prospects For An Arm Server Insurrection [Электронный ресурс]. – Режим доступа: https://www.nextplatform.com/2021/05/03/the-prospects-for-an-arm-server-insurrection/, свободный (дата обращения: 01.09.2021).

4. /NXCOMPAT (Compatible with Data Execution Prevention) [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/cpp/build/reference/nxcompat-compatible-with-data-execution-prevention?view=vs-2019, свободный (дата обращения: 01.09.2021).

5. How to Configure Memory Protection in Windows XP SP2 [Электронный ресурс]. – Режим доступа: https://technet.microsoft.com/en-us/library/cc700810.aspx , свободный (дата обращения: 01.09.2021).

6. Buchanan E. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. / E. Buchanan, R. Roemer, H. Shacham [at al] // Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS). – 2008.

7. /DYNAMICBASE (Use address space layout randomization) [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/cpp/build/reference/dynamicbase-use-address-space-layout-randomization?view=vs-2019, свободный (дата обращения: 01.09.2021).

8. Kernel address space layout randomization [Электронный ресурс] – Режим доступа: https://lwn.net/Articles/569635/ (дата обращения: 31.08.2018).

9. Shacham H. On the effectiveness of address-space randomization / H. Shacham, M. Page, B. Pfaff [at al] // Proceedings of the 11th ACM conference on Computer and Communications Security (CCS). – 2004.

10. Bennett J. The Number of the Beast [Электронный ресурс] / J. Bennett, Y. Lin, T. Haq. – Режим доступа: https://www.fireeye.com/blog/threat-research/2013/02/the-number-of-the-beast.html, свободный (дата обращения: 01.09.2021).

11. SolarWinds releases security advisory after Microsoft discovers vulnerability [Электронный ресурс]. – Режим доступа: https://www.zdnet.com/article/solarwinds-releases-security-advisory-after-microsoft-says-customer-targeted-through-vulnerability/, свободный (дата обращения: 01.09.2021).

12. Варновский Н.П. Современные методы обфускации программ: сравнительный анализ и классификация [Электронный ресурс] / Н.П. Варновский, В.А. Захаров [и др.] // Известия ЮФУ. Технические науки. – 2007. – № 1. – Режим доступа: https://cyberleninka.ru/article/n/sovremennye-metody-obfuskatsii-programm-sravnitelnyy-analiz-i-klassifikatsiya (дата обращения: 26.10.2021).

13. Нурмухаметов А.Р. Описание подхода к разработке обфусцирующего компилятора / А.Р. Нурмухаметов, Ш.Ф. Курмангалеев [и др.] // Труды ИСП РАН. – 2012. – Т. 23. – Вып. 3. – С. 67–76.

14. 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.

15. 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.

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

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

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

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

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

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

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

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

24. 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.

25. Шудрак М.О. Методика декомпиляции бинарного кода и её применение в сфере информационной безопасности / М.О. Шудрак, И.А. Лубкин // Материалы

II Всероссийской молодежной конференции «Перспектива – 2012». – Таганрог, 2012. – С. 197–202.

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

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

28. Гласс Р. Факты и заблуждения профессионального программирования / Р. Гласс. – СПб.: Символ-Плюс, 2007. – 240 с.

29. ГОСТ 56939-2016. Защита информации. Разработка безопасного программного обеспечения. Общие требования. – М.: Стандартинформ, 2016. – 20 с.

30. ГОСТ 51275-2006. Защита информации. Объект информатизации. Факторы, воздействующие на информацию. Общие положения. – М.: Стандартинформ, 2014. – 8 с.

31. ГОСТ Р 56546-2015. Защита информации. Уязвимости информационных систем. Классификация уязвимостей информационных систем. – М.: Стандартинформ, 2018. – 8 с.

32. X86-64 Buffer Overflow Exploits and the Borrowed Code Chunks Exploitation Technique [Электронный ресурс]. – 2005. – Режим доступа: http://users.suse.com/%7Ekrahmer/no- nx.pdf (дата обращения: 18.08.2018).

33. Launching Return-Oriented Programming Attacks Against Randomized Relocatable Executables / L. Liu [at al] // Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. – Washington, DC, USA: IEEE Computer Society, 2011. – P. 37–44. – (TRUSTCOM ’11). – DOI: 10.1109/TrustCom.2011.9.

34. Evtyushkin D. Jump over ASLR: Attacking Branch Predictors to Bypass ASLR / D. Evtyushkin, D. Ponomarev, N. Abu-Ghazaleh // The 49th Annual

IEEE/ACM International Symposium on Microarchitecture. – Taipei, Taiwan: IEEE Press, 2016. – 40:1–40:13. – (MICRO-49).

35. Hund R. Practical Timing Side Channel Attacks Against Kernel Space ASLR / R. Hund, C. Willems, T. Holz // Proceedings of the 2013 IEEE Symposium on Security and Privacy. – Washington, DC, USA: IEEE Computer Society, 2013. – PP. 191–205. – (SP ’13). – DOI: 10.1109/SP.2013.23.

36. Gu Y. Derandomizing Kernel Address Space Layout for Memory Introspection and Forensics / Y. Gu, Z. Lin // Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. – New Orleans, Louisiana, USA: ACM, 2016. — PP. 62–72. – (CODASPY ’16). – DOI: 10.1145/2857705.2857707.

37. Surgically Returning to Randomized Lib(C) / G. F. Roglia [at al] // Proceedings of the 2009 Annual Computer Security Applications Conference. – Washington, DC, USA: IEEE Computer Society, 2009. – PP. 60–69. – (ACSAC ’09). – DOI: 10.1109/ACSAC.2009.16.

38. SystemV Application Binary Interface. AMD64 Architecture Processor Supplement. Version 1.0 [Электронный ресурс] / Под редакцией H.J.Lu, Michael Matz, Milind Girkar [at al]. – January 28, 2018. – Режим доступа: https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-1.0.pdf

39. Вишняков А.В. Классификация *RoP*-гаджетов / А.В. Вишняков // Труды ИСП РАН. – 2016. – Т. 28. – Вып. 6. – С. 27–36. DOI: 10.15514/ISPRAS-2016-28(6)-2.

40. Shanbhogue Vedvyas. Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity / Vedvyas Shanbhogue, Deepak Gupta and Ravi Sahita // Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '19). Association for Computing Machinery, New York, NY, USA. – 2019. – Article 8, 1–11. DOI: https://doi.org/10.1145/3337167.3337175

41. Intel® 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4 [Электронный ресурс]. – Режим

доступа: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf

42. Intel Launches World’s Best Processor for Thin-and-Light Laptops: 11th Gen Intel Core [Электронный ресурс]. – Режим доступа: https://www.intel.com/content/www/us/en/newsroom/news/11th-gen-tiger-lake-evo.html

43. RAP:RIP ROP 2015 [Электронный ресурс]. – Режим доступа: https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf

44. Cowan C. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks / C. Cowan [at al] // USENIX security symposium. – Vol. 98. – San Antonio, TX, 1998. – PP. 63–78.

45. Koo, Z.Z. Analysis of ROP attack on grsecurity. PaX linux kernel security variables / Z.Z. Koo, Zakiah Ayop, Z.Z. Abidin // International Journal of Applied Engineering Research, 2017. – no. 12. – PP. 13179–13185.

46. Bypassing PaX ASLR protection [Электронный ресурс]. – Режим доступа: http://phrack.org/issues/59/9.html (дата обращения: 26.08.2018).

47. Иванников В. Реализация запутывающих преобразований в компиляторной инфраструктуре LLVM / В. Иванников [и др.] // Труды ИСП РАН. – 2014. – Т. 26. – Вып. 1. – С. 327–342.

48. Нурмухаметов А.Р. Применение компиляторных преобразований для противодействия эксплуатации уязвимостей программного обеспечения / А.Р. Нурмухаметов [и др.] // Труды ИСП РАН. – 2014. – Т. 26. – Вып. 3. – С. 113–126. DOI: 10.15514/ISPRAS-2014-26(3)-6.

49. ИСП Обфускатор. Технология запутывания кода для защиты от эксплуатации уязвимостей [Электронный ресурс]. – Режим доступа: https://www.ispras.ru /technologies/isp_obfuscator/

50. Нурмухаметов А.Р. Мелкогранулярная рандомизация адресного пространства программы при запуске / А.Р. Нурмухаметов [и др.] // Труды ИСП РАН. – 2017. – Т. 29. – Вып. 6. – С. 163–182. DOI: 10.15514/ISPRAS-2017-29(6)-9.

51. Crane S. Code randomization: Haven’t we solved this problem yet? Cybersecurity Development (SecDev) / S. Crane [at al] // IEEE, 2016.

52. Conti M. Securing the tor browser against de-anonymization exploits / M. Conti [at al] // PoPETs. – 2016. – no. 4. – PP. 454–469.

53. Williams-King D. Shuffler: Fast and deployable continuous code re-randomization / D. Williams-King [at al] // Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. – 2016. – PP. 367–382.

54. Kangjie Lu. How to Make ASLR Win the Clone Wars: Runtime Re-Randomization / Lu Kangjie [at al] // Proceedings of the 23rd Annual Network and Distributed System Security Symposium. – 2016.

55. Onarlioglu K. G-Free: Defeating return-oriented programming through gadget-less binaries / K. Onarlioglu [at al] // Proceedings of ACSAC: M. Franz and J. McDermott, Eds. – ACM Press, 2010. – PP. 49–58.

56. Jinku Li. Defeating return-oriented rootkits with «return-less» kernels / Li Jinku [at al] // Proceedings of EuroSys, 2010, edited by G. Muller. – ACM Press. – PP. 195–208.

57. Sullivan Dean. Execution Integrity with In-Place Encryption / Dean Sullivan [at al] // Execution Integrity with In-Place Encryption. arXiv preprint arXiv:1703.02698 (2017).

58. Bittau A. Hacking Blind / A. Bittau [at al] // Proceedings of the 2014 IEEE Symposium on Security and Privacy. – Washington, DC, USA. – IEEE Computer Society, 2014. – C. 227–242. –(SP ’14). – DOI: 10.1109/SP.2014.22.

59. Zhang T. SeBROP: blind ROP attacks without returns / T. Zhang [at al]. – Front. Comput. Sci. 16, 164818 (2022).

60. Göktas E. Position-Independent Code Reuse: On the Effectiveness of ASLR in the Absence of Information Disclosure. / E. Göktas [at al]. – IEEE European Symposium on Security and Privacy (EuroS&P), 2018. – PP. 227–242.

61. Bosman E., BOS H. Framing Signals – A Return to Portable Shellcode. SP '14 / E. Bosman, H. BOS // Proceedings of the 2014. – IEEE Symposium on Security and Privacy. – May 2014. – PP. 243–258.

62. Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks / H. Hu [at al] // IEEE Symposium on Security and Privacy (SP). – 05.2016. – PP. 969–986. – DOI: 10.1109/SP.2016.62.

63. Dullien T.F. Weird machines, exploitability and provable unexploitability / T.F. Dullien // IEEE Transactions on Emerging Topics in Computing. – 2017.

64. Ispoglou K.K., Block Oriented Programming: Automating Data-Only Atacks / K.K. Ispoglou [at al] // ACM SIGSAC Conference on Computer and Communications Security (CCS ’18), October 15–19, 2018, Toronto, ON, Canada. – ACM, New York, NY, USA. – 16 pages.

65. Guo Y., Function-Oriented Programming: A New Class of Code Reuse Attack in C Applications / Y. Guo [at al] // IEEE Conference on Communications and Network Security (CNS). – 2018. – PP. 1–9.

66. Kirsch J. Dynamic Loader Oriented Programmint on Linux / Kirsch J. [at al] // Proceedings of Reversing and Offensive-oriented Trends Symposium. – 2017. – PP. 1–13.

67. Jump-oriented Programming: A New Class of Code-reuse Attack / T. Bletsch [at al] // Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. – Hong Kong, China. – ACM, 2011. – PP. 30–40 (ASIACCS ’11).

68. Sadeghi A., Tiny Jump-oriented programming Atack / A. Sadeghi, F. Aminmansoure, H. Shahriari // Proceedings of the 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC). – 2015. – PP. 52–57.

69. Sadeghi A. Pure-Call Oriented Programming (PCOP): chaining the gadgets using call instructions / A. Sadeghi [at al] // J Comput Virol Hack Tech 14. – 2018. – PP. 139–156.

70. Schuster Felix & Tendyck. Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications / Felix Schuster & Tendyck [at al] // 2015. 10.1109/SP.2015.51.

71. Crane Stephen & Volckaert. It's a TRaP: Table Randomization and Protection against Function-Reuse Attacks / Stephen Crane & Volckaert [at al] // 2015. 10.1145/2810103.2813682.

72. Backes M. Oxymoron: Making fine-grained memory randomization practical by allowing code sharing / M. Backes, S. Nurberger // Proceedings of the 23rd USENIX Security Symposium. – 2014. – PP. 433–447.

73. Bigelow D. Timely rerandomization for mitigating memory disclosures / D. Bigelow, T. Hobson, R. Rudd [et al] // Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. – 2015. – PP. 268–279.

74. Davi L. Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and ARM / L. Davi, A. Dmitrienko, S. Nurnberger [at al] // 8th ACM Symposium on Information, Computer and Communications Security. – 2013.

75. Szor P., Ferrie P. Hunting for metamorphic [Электронный ресурс] / P. Szor, P. Ferrie // Symantec Security Response. – 2003. Режим доступа:ttp://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf (дата обращения: 01.09.2021).

76. TIS Committee Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification [Электронный ресурс]. – 1995. – Режим доступа: http://refspecs.linuxbase.org/elf/elf.pdf (дата обращения: 15.03.2019).

77. libelf [Электронный ресурс]. – Режим доступа: https://sourceforge.net/p/elftoolchain/wiki/libelf/, свободный (дата обращения: 01.09.2021).

78. Exception handling tables [Электронный ресурс]. – Режим доступа: http://itanium-cxx-abi.github.io/cxx-abi/exceptions.pdf, свободный (дата обращения: 01.09.2021).

79. DWARF Debugging Information Format. Version 5 [Электронный ресурс]. – Режим доступа: https://dwarfstd.org/doc/DWARF5.pdf, свободный (дата обращения: 01.09.2021).

80. AMD64 Architecture Programmers Manual, Volume 3 [Электронный ресурс]. – Режим доступа: https://www.amd.com/system/files/TechDocs/24594.pdf, свободный (дата обращения: 01.09.2021).

81. AMD64 Architecture Programmers Manual, Volume 2 [Электронный ресурс]. – Режим доступа: https://www.amd.com/system/files/TechDocs/24593.pdf, свободный (дата обращения: 01.09.2021).

82. Sebesta R. Concepts of programming languages / R. Sebesta // 5th edition. – Boston, Addison-Wesly, 2002.

83. Cooper K. Engineering a compiler / K. Cooper, L. Torczon // Amsterdam: Elseveir, 2012.

84. Turing A. On Computable Numbers, with an Application to the Entscheidungsproblem / A. Turing // Proceedings of the London Mathematical Society. – Vol. s2-42, Iss. 1. – London Mathematical Society, 1937. – PP. 230–265.

85. Aho A. Compilers: principles, techniques and tools / A. Aho, M. Lam [at al]. – Addison-Wesly, 2006.

86. IDA Pro – интерактивный дизассемблер [Электронный ресурс]. – Режим доступа: http://www.idasoft.ru/idapro/ (дата обращения: 01.09.2021).

87. Ghidra [Электронный ресурс]. – Режим доступа: https://www.nsa.gov/resources/everyone/ghidra (дата обращения: 01.09.2021).

88. Heller M. Despite reservations about NSA's Ghidra, experts see value [Электронный ресурс] / M. Heller. – Режим доступа: https://searchsecurity.techtarget.com/news/252459574/Despite-reservations-about-NSAs-Ghidra-experts-see-value.

89. radare2 [Электронный ресурс]. – Режим доступа: https://www.radare.org/n/radare2.html, свободный (дата обращения: 01.09.2021).

90. Встроенный отладчик и дизассемблер в Linux gdb [Электронный ресурс]. – Режим доступа: www.gnu.org/s/gdb/ (дата обращения: 01.09.2021).

91. Интерактивный отладчик OllyDbg [Электронный ресурс]. – Режим доступа: www.ollydbg.de (дата обращения: 01.09.2021).

92. Интерактивный отладчик WinDbg [Электронный ресурс]. – Режим доступа: www.windbg.org (дата обращения: 01.09.2021).

93. fde64 [Электронный ресурс]. – Режим доступа: https://github.com/GiveMeZeny/fde64, свободный (дата обращения: 01.09.2021).

94. IDA 7.0 free [Электронный ресурс]. – Режим доступа: https://github.com/AngelKitty/IDA7.0, свободный (дата обращения: 01.09.2021).

95. Cppcheck. A tool for static C/C++ code analysis [Электронный ресурс]. – Режим доступа: https:// http://cppcheck.net/, свободный (дата обращения: 01.09.2021).

96. asn1c [Электронный ресурс]. – Режим доступа: https://github.com/vlm/asn1c, свободный (дата обращения: 01.09.2021).

97. Shacham Hovav. The Geometry of Innocent Flash on the Bone: Return-into-libc without Function Calls (on the x86) / Hovav Shacham // ACM Conference on Computer and Communications Security (CCS). – Proceedings of CCS, 2007. – PP. 552–561.

98. Permutation conditions [Электронный ресурс]. – Режим доступа: https://z0mbie.dreamhosters.com/pcond.txt (дата обращения 01.09.2021).

99. Репозиторий с исходным кодом библиотеки eXtended Disassembler Engine (version 1.02) [Электронный ресурс]. – Режим доступа: https://github.com/nimrood/xde (дата обращения 01.09.2021).

100. Wagle Perry, Cowan Crispin. StackGuard: Simple Stack Smash Protection for GCC, GCC Developers Summit, 2003 [Электронный ресурс] / Perry Wagle, Crispin Cowan. – Режим доступа: https://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf, свободный (дата обращения: 01.09.2021).

101. Инструмент ROPgadget. Репозиторий с исходным кодом [Электронный ресурс]. – Режим доступа: https://github.com/JonathanSalwan/ROPgadget (дата обращения 01.09.2021).

102. coremark [Электронный ресурс]. – Режим доступа: https://github.com/eembc/coremark, свободный (дата обращения: 01.09.2021).

103. gfree/README.md [Электронный ресурс]. – Режим доступа: https://github.com/pagabuc/gfree/blob/master/README.md, свободный (дата обращения 01.09.2021).

104. Return oriented programming. Собираем exploit по кусочкам [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/255519/, свободный (дата обращения 01.09.2021).

ПРИЛОЖЕНИЕ 1. СПИСОК ПРОАНАЛИЗИРОВАННЫХ УЯЗВИМОСТЕЙ

№	CVE	ПО, содержащее уязвимость, и его окружение	Найден ли уязвимый экземпляр ПО	Время от внесения до устранения уязвимости, месяцев	Доступен ли публично эксплойт, гаджеты из его состава	Ссылки, источники, особые условия
1.	CVE-2010-2883	icucnv34.dll (8.0, 8.3.0), icucnv36.dll (9.5.3)	Неактуально	Неактуально	Нет	https://www.securitylab.ru/analytics/471746.php
2.	CVE-2011-1938	Archlinux, binary php-5.3.6 x86-32	Неактуально	<1	Нет	http://shell-storm.org/blog/PHP-5.3.6-Buffer-Overflow-PoC-ROP/
3.	CVE-2011-2462	icucnv36.dll (9.5.3)	Неактуально	1	Да	https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2011-2462
4.	CVE-2013-0641	AcroForm.api (10.0.7, 10.1.2)	Неактуально	5	Да	https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2013-0641
5.	CVE-2013-2729	AcroRd.dll (9.5.3)	Неактуально	3	Нет	https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2013-2729
6.	CVE-2013-4988	IcoFX 2.6	Да	1	Да	https://sploitus.com/exploit?id=EDB-ID:49959
7.	CVE-2014-0569	Adobe Flash Player 11.2.202.223	Неактуально	1	Нет	https://www.securitylab.ru/analytics/471746.php
8.	CVE-2018-4990	EScript.api (18.01120038)		1	Нет	https://habr.com/ru/company/eset/blog/358688/
9.	CVE-2018-5767	Tenda’s AC15 router ARM	Неактуально	1	Нет	https://fidusinfosec.com/remote-code-execution-cve-2018-5767/
10.	CVE-2017-0240	Windows 10 Pro 1703 (10.0.15063) и браузер Microsoft Edge (40.15063.0.0)	Нет	2	Нет	https://habr.com/ru/company/dsec/blog/455594/
11.	CVE-2017-0781	Android, iOS, Windows и Linux	Неактуально	1	Да	
12.	CVE-2017-0782			1	Да	
13.	CVE-2017-14315	iOS	Неактуально	1	Да	
14.	CVE-2017-1000251	linux v. 2.6.32 – 4.14	Неактуально	2	Нет	

№	CVE	ПО, содержащее уязвимость, и его окружение	Найден ли уязвимый экземпляр ПО	Время от внесения до устранения уязвимости, месяцев	Доступен ли публично эксплойт, гаджеты из его состава	Ссылки, источники, особые условия
15.	CVE-2019-0940	Уточняется	Неактуально	3	Нет	https://blog.exodusintel.com/2019/05/19/pwn2own-2019-microsoft-edge-renderer-exploitation-cve-2019-9999-part-1/
16.	CVE-2019-18683	Ubuntu Server 18.04	Неактуально	0 (исправлена до опубликования)	Да, см. ниже	https://habr.com/ru/company/pt/blog/491756/
17.	CVE-2019-14363	Netgear N600 WiFi Dual Band Router (WNDR3400v3) V1.0.1.18 – V1.0.1.24	Неактуально	1	Нет	https://github.com/reevesrs24/CVE/blob/master/Netgear_WNDR2400v3/upnp_stack_overflow/upnp_stack_overflow.md
18.	CVE-2020-1350	Windows Server versions 2003 to 2019	Неактуально	204	Нет	https://research.checkpoint.com/2020/resolving-your-way-into-domain-admin:-exploiting-a-17-year-old-bug-in-windows-dns-servers/
19.	CVE-2020-1380	Internet Explorer 11 and Windows 10 build 18363 x64		0 (исправлена до опубликования)	Нет	https://securelist.com/ie-and-windows-zero-day-operation-powerfall/97976/
20.	CVE-2020-0796	Windows 10 1903, Windows 10 1909, Windows Server 1903 и Windows Server 1909	Неактуально	1	Нет	https://ricercasecurity.blogspot.com/2020/04/ill-ask-your-body-smbghost-pre-auth-rce.html https://xakep.ru/2020/04/02/smbghost-pocs/
21.	CVE-2020-16898	Windows версии 10 1803, 10 1809, 10 1909, 10 1709, 10 1903, 10 2004 Windows Server версии 2019, 2019 1909, 2019 1903, 2019 2004	Неактуально	2	Нет	https://www.securitylab.ru/vulnerability/512993.php
22.	CVE-2019-7051	Adobe Acrobat Pro DC 2019.010.20069	Да	3	Нет	https://nvd.nist.gov/vuln/detail/CVE-2019-7051
23.	CVE-2020-8423	TP-LINK TL-WR841N V10 router	Неактуально	2	Нет	https://ktln2.org/2020/03/29/exploiting-mips-router/
24.	CVE-2020-0022	Android-8.0-9.0	Неактуально	<1	Нет	https://insinuator.net/2020/04/cve-2020-0022-an-android-8-0-9-0-bluetooth-zero-click-rce-bluefrag/

№	CVE	ПО, содержащее уязвимость, и его окружение	Найден ли уязвимый экземпляр ПО	Время от внесения до устранения уязвимости, месяцев	Доступен ли публично эксплойт, гаджеты из его состава	Ссылки, источники, особые условия
25.	CVE-2020-0041	Android-8.0-9.0	Неактуально	<1	Нет	https://labs.bluefrostsecurity.de/blog/2020/03/31/cve-2020-0041-part-1-sandbox-escape/
26.	CVE-2020-6449	x64 1.6 «Смоленск» (Astra Linux)	Да	<1	Нет	https://github.com/github/securitylab/tree/main/SecurityExploits/Chrome/blink/CVE-2020-6449 https://securitylab.github.com/research/CVE-2020-6449-exploit-chrome-uaf
27.	CVE-2020-17382	x64 Windows 10 version, 2004	Неактуально	<1	Да, см. ниже	https://www.matteomalvica.com/blog/2020/09/24/weaponizing-cve-2020-17382/
28.	CVE-2020-11896	ARMv9	Неактуально	1	Нет	https://www.jsof-tech.com/wp-content/uploads/2020/06/JSOF_Ripple20_Technical_Whitepaper_June20.pdf
29.	CVE-2020-12351	linux v. 5.8.0.48	Да	3	Да	https://github.com/google/security-research/blob/master/pocs/linux/bleedingtooth
30.	CVE-2020-14871	X86 Oracle Solaris 11	Неактуально	5	Нет	https://www.tenable.com/blog/cve-2020-14871-critical-buffer-overflow-in-oracle-solaris-exploited-in-the-wild-as-zero-day
31.	EDB-ID: 48264	i386 10-Strike Network Inventory Explorer 9.03 Win8.1 x64 – Build 9600	Да	Н/д	Нет	https://www.exploit-db.com/exploits/48264
32.	EDB-ID: 48840 CVE-2018-6892	i386 CloudMe 1.11.2 Windows 10 (x64) – 10.0.19041 Build 19041	Да	6	Нет	https://www.exploit-db.com/exploits/48840
33.	EDB-ID 44820	Sony Playstation 3 (PS3) 4.82	Неактуально	Н/д	Нет	https://www.exploit-db.com/exploits/44820
34.	EDB-ID 44426	pms_0.42-1 + b2_i386	Да	Н/д	Нет	https://www.exploit-db.com/exploits/44426
35.	EDB-ID 4433	Crashmail 1.6	Да	Н/д	Нет	https://www.exploit-db.com/exploits/44331
36.	CVE-2020-1380	Internet Explorer, for Windows 10 2004, KB4565503 and previous	Нет	0 (исправлена до опубликования)	Нет	https://googleprojectzero.blogspot.com/p/rca-cve-2020-1380.html

№	CVE	ПО, содержащее уязвимость, и его окружение	Найден ли уязвимый экземпляр ПО	Время от внесения до устранения уязвимости, месяцев	Доступен ли публично эксплойт, гаджеты из его состава	Ссылки, источники, особые условия
37.	CVE-2021-22555	linux v. 5.8.0.48	Да	3	Да	https://github.com/google/security-research/blob/master/pocs/linux/cve-2021-22555
38.	CVE-2021-30734	Safari (< 14.1.1)	Да	0	Да	https://blog.ret2.io/2021/06/02/pwn2own-2021-jsc-exploit/
39.	CVE-2021-35211	Serv-U version 15.2.3.717	Да	1	Да	https://github.com/BishopFox/CVE-2021-35211
40.	CVE-2022-25636	linux v. 5.13.0-30	Да	0	Да	https://github.com/Bonfee/CVE-2022-25636
41.	CVE-2022-0995	Ubuntu kernel 5.13.0-37-generic	Да	0	Да	https://github.com/Bonfee/CVE-2022-0995
42.	CVE-2022-1015	Ubuntu kernel 5.13.0-37-generic	Да	0	Да	https://github.com/pqlx/CVE-2022-1015
43.	Нет	Синтетический пример	Нет	Неприменимо	Нет	https://habr.com/ru/post/255519/ gcc -fno-stack-protector
44.	Нет	Синтетический пример	Да	Неприменимо	Да, см. ниже	https://habr.com/ru/post/479184/
45.	Нет	Синтетический пример	Да	Неприменимо	Да, см. ниже	https://vc.ru/u/399338-codeby-net/103531-rop-cepochki-i-gadzhety-uchimsya-razrabatyvat-eksployty

ПРИЛОЖЕНИЕ 2. ГАДЖЕТЫ ИЗ СОСТАВА ПРОАНАЛИЗИРОВАННЫХ УЯЗВИМОСТЕЙ

Найденные гаджеты, используемые в проанализированных уязвимостях, приведены в таблице далее. Каждый приведенный в таблице гаджет занимает одну строку. Инструкции процессора в строке представлены в нотации Intel.

№	Эксплойт	Архитектура процессора	Вид цепочек	Выявленные гаджеты
1.	Для синтетических примеров	i386	RoP	pop edx ; xor eax, eax ; pop edi ; ret mov dword ptr [edx], edi ; pop esi ; pop edi ; ret pop edx ; xor eax, eax ; pop edi ; ret mov dword ptr [edx], eax ; ret pop edx ; xor eax, eax ; pop edi ; ret pop ecx ; add al, 0xa ; ret pop ebx ; ret inc eax ; ret inc esi ; int 0x80 Pop ecx;pop edx;ret xor eax,eax; pop ebp; ret inc eax; ret pop ebx; ret int 0x80

2.	EDB-ID 48264	i386	RoP	POP EBX # RETN
				POP EAX # RETN
				NEG EAX # RETN
				add edi, eax # add eax, 41140e0a # ret
				POP EAX # RETN
				NEG FFFFFFfF = 0x01
				NEG EAX # RETN
				DEC EAX # RETN
				XCHG EAX,ECX # RETN
				XOR ESI,ESI # RETN
				ADD ESI,EDI # ADD AL,0A # RETN
				mov [esi], cl # adc al, 41 # ret
				POP EAX # RETN
				ADD ESI,EAX # INC EBP # RETN
				POP EAX # RETN
				NEG FF7E5E98 = 0081A168 = PTR Kernel32.LoadLibraryA
				NEG EAX # RETN
				mov eax,dword ptr ds:[eax]
				XCHG EAX,ESI # RETN
				POP ECX # RETN
				RETN
				POP EDI # RETN
				pushad # ret

№	Экс-плойт	Архитектура процессора	Вид це-почек	Выявленные гаджеты

				XCHG EAX,EBP # RETN
				POP EAX # RETN
				NEG FFFFFB5E = 0x4A2
				NEG EAX # RETN
				add edi, eax # add eax, 41140e0a # ret
				POP EAX # RETN
				NEG FFFFFFfF = 0x01
				NEG EAX # RETN
				DEC EAX # RETN
				XCHG EAX,ECX # RETN
				XOR ESI,ESI # RETN
				ADD ESI,EDI # ADD AL,0A # RETN
				mov [esi], cl # adc al, 41 # ret
				POP EAX # RETN
				ADD ESI,EAX # INC EBP # RETN
				DEC EBP # RETN
				POP EAX # RETN
				NEG FF7E5EB8 = 0081A148 = PTR Ker-nel32.GetProcAddr
				NEG EAX # RETN
				mov eax,dword ptr ds:[eax]
				XCHG EAX,ESI # RETN
				XCHG EAX,ECX # RETN
				XCHG EAX,EBP # RETN
				XCHG EAX,ECX # RETN
				POP EDX # RETN
				RETN
				POP EDI # RETN
				ret 0C
				pushad # ret
				XCHG EAX,EDX # RETN
				POP EAX # RETN
				NEG 0xFFFFFBC8 = 0x438
				NEG EAX # RETN
				add edi, eax # add eax, 41140e0a # ret
				POP EAX # RETN
				NEG FFFFFFfF = 0x01
				NEG EAX # RETN
				DEC EAX # RETN
				XCHG EAX,ECX # RETN
				XOR ESI,ESI # RETN
				ADD ESI,EDI # ADD AL,0A # RETN
				mov [esi], cl # adc al, 41 # ret
				POP EAX # RETN
				ADD ESI,EAX # INC EBP # RETN
				POP ECX # RETN
				POP EDI # RETN
				XCHG EAX,EDX # RETN
				XCHG EAX,ESI # RETN
				POP EDI # RETN

№	Экс-плойт	Архитектура процессора	Вид це-почек	Выявленные гаджеты

				pushad # ret
				XCHG EAX,ESI # RETN
				XCHG EAX,EDX # RETN
				POP EAX # RETN
				NEG 0xFFFFFC09 = 0x3F7
				NEG EAX # RETN
				add edi, eax # add eax, 41140e0a # ret
				POP EAX # RETN
				NEG FFFFFFfF = 0x01
				NEG EAX # RETN
				DEC EAX # RETN
				XCHG EAX,ECX # RETN
				XOR ESI,ESI # RETN
				ADD ESI,EDI # ADD AL,0A # RETN
				mov [esi], cl # adc al, 41 # ret
				POP EAX # RETN
				ADD ESI,EAX # INC EBP # RETN
				POP ECX # RETN
				POP EDI # RETN
				XCHG EAX,EDX # RETN
				XCHG EAX,ESI # RETN

3.	EDB-ID 44426	i386	RoP	pop edx ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret pop eax ; ret mov dword ptr [edx], eax ; pop ebx ; pop ebp ; ret pop edx ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret pop eax ; ret mov dword ptr [edx], eax ; pop ebx; pop ebp ; ret pop edx ; pop ebx ; pop esi ; pop edi; pop ebp ; ret xor eax, eax ; pop esi ; pop ebp ; ret mov dword ptr [edx], eax ; pop ebx ;pop ebp ; ret pop ebx ; pop esi ; pop edi ; ret pop ecx ; ret pop edx ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret xor eax, eax ; pop esi ; pop ebp ; ret inc eax ; ret int 0x80
4.	EDB-ID 48264	i386	RoP	POP EDX # RETN MOV EAX,DWORD PTR DS:[EDX] # RETN XCHG EAX,ESI # ADD AL,BYTE PTR DS:[ECX] # RETN POP EBP # RETN push esp # ret 0x04 POP EAX # RETN NEG EAX # RETN POP EBX # RETN INC EBX # RETN ADD EBX,EAX # RETN POP EDX # RETN NEG EDX # RETN POP ECX # RETN

№	Экс-плойт	Архитектура процессора	Вид це-почек	Выявленные гаджеты
				POP EDI # RETN RETN POP EAX # RETN PUSHAD # RETN
5.	EDB-ID 4433	i386	RoP	pop eax; ret pop edx; ret mov dword ptr [edx], eax; ret pop eax; ret pop edx; ret mov dword ptr [edx], eax; ret xor eax, eax; ret pop edx; ret mov dword ptr [edx], eax; ret pop ebx; ret pop ecx; push cs; adc al, 0x41; ret pop edx; ret pop eax; ret neg eax; ret int 0x80; ret

6.	CVE-2013-4988	i386	RoP/JoP	POP ECX # RETN
				ADD ECX,EBP # RETN
				POP EAX # RETN
				ADD ECX,EAX # RETN
				MOV EAX,ECX # POP ESI # RETN
				DEC EAX # RETN
				POP EBX # RETN
				POP EDI # RETN
				add ecx, dword ptr [eax] 0x406d81 #jmp dword ptr [ecx]
				JMP EDI
				pop ebp # or byte ptr [ebx - 0x781703bb], cl # jmp edi
				jmp dword ptr [ebp - 0x71]

7.	CVE-2019-18683	amd64	RoP/ JoP	PUSH RDI; POP RSP; pop rbp;or eax edx; RET POP R15; RET POP RDI;RET JMP R15
8.	CVE-2020-6449	amd64	RoP	mov rax,QWORD PTR [rdi + 0x20]; mov rsi,QWORD PTR [rdi + 0x98]; mov rdx,QWORD PTR [rdi + 0xa0]; add rdi, 0x28
9.	CVE-2020-12351	amd64	RoP/ JoP	PUSH RSI; ADD BYTE PTR RBX+41,BL; POP RSP; POP RBP; RET POP RAX; RET POP RDI; RET JMP RAX
10.	CVE - 2020-17382	amd64	RoP	pop rcx; ret mov cr4, ecx; ret
11.	CVE-	amd64	RoP	ret

№	Эксплойт	Архитектура процессора	Вид цепочек	Выявленные гаджеты
	2021-30734			pop rdi; …; ret pop rsi; …; ret pop rdx; …; ret Цели: ExecutableAllocator::allocate(arg1,arg2,arg3) и memcpy
12.	CVE-2022-25636	amd64	RoP	pop rdi; ret xor dh, dh mov rax, rdi pop r12; pop r13; pop r14; pop r15; retn pop rax; pop rdi; swapgs; iret Примечание: утечка адреса базы ядра через ioctl(fd, SIOCGIFHWADDR, leak)

13.	CVE-2021-22555	amd64	RoP/ JoP	push rax ; jmp qword ptr [rcx]
				pop rsp ; pop rbx ; ret
				enter 0, 0 ; pop rbx ; pop r14 ; pop rbp ; ret
				mov qword ptr [r14], rbx ; pop rbx ; pop r14 ; pop rbp ; ret
				push qword ptr [rbp + 0x25] ; pop rbp ; ret
				mov rsp, rbp ; pop rbp ; ret
				pop rdx ; ret
				pop rsi ; ret
				pop rdi ; ret
				pop rbp ; ret
				mov rdi, rax ; jne 0xffffffff81238396 ; pop rbp ; ret
				cmp rdx, 1 ; jne 0xffffffff8152831d ; pop rbp ; ret
				push rsi ; jmp qword ptr [rsi + 0x39]
				pop rsp ; ret
				add rsp, 0xd0 ; ret
				enter 0, 0 ; pop rbx ; pop r12 ; pop rbp ; ret
				mov qword ptr [r12], rbx ; pop rbx ; pop r12 ; pop rbp ; ret
				push qword ptr [rbp + 0xa] ; pop rbp ; ret
				mov rsp, rbp ; pop rbp ; ret
				pop rcx ; ret
				mov rdi, rax ; jne 0xffffffff81557888 ; xor eax, eax ; ret
				cmp rcx, 4 ; jne 0xffffffff810724c0 ; pop rbp ; ret
				Примечание: утечка адреса базы ядра за счет чтения памяти поврежденного объекта после UAF

14.	CVE-2021-35211	amd64	RoP/ JoP	mov rsp, r11 ; pop r14 ; ret int 3; ret 0 nop ; ret jmp rax jmp qword ptr [rax] call qword ptr [rbx] push rax; retn push rsi; retn push rbp ; pop rax ; add byte ptr [rax], al ; ret push rax ; pop rbx ; ret

№	Экс-плойт	Архитектура процессора	Вид це-почек	Выявленные гаджеты

				push rbp ; add dword [rax], eax ; add rsp, 0x20 ; pop rbx; ret
				pop rbp ; pop rbx ; ret
				pop rsp ; pop rdi; ret
				pop rdx ; ret
				pop rbx ; ret
				pop rax ; ret
				pop rcx ; ret
				pop rdi ; ret
				pop r8 ; ret
				pop r15 ; ret
				mov rax, r11 ; ret
				mov rax, rdx ; ret
				mov rcx, rax ; cvttsd2si rax, xmm0 ; add rax, rcx ; add rsp, 0x28 ; ret
				mov rax, qword ptr [rbx + 0x20] ; add rsp, 0x20 ; pop rbx ; ret
				mov rax, qword ptr [rax] ; add rsp, 0x20 ; pop rbx ; ret
				mov qword ptr [rbx], rax ; add rsp, 0x20 ; pop rbx ; ret
				mov [rcx], rax
				mov [rdx], rax
				mov qword ptr [rdx], rax ; mov rax, rdx ; ret
				xchg rax, r9 ; adc al, 0 ; add rsp, 0x38 ; ret
				xchg eax, ebx ; retf 0x56
				xchg eax, ebx ; ret
				xchg eax, esp ; ret
				add rax, rcx ; ret
				add rax, rdx ; ret
				adc al, 0 ; add rsp, 0x28 ; ret
				sub rax, 8 ; ret
				xor al, al

15.	CVE-2022-0995	amd64	RoP	push rsi ; mov edx, 0x415b00c3 ; pop rsp ; pop rbp ; ret pop r12; pop r15; ret pop rdi; ret xor dh, dh; ret mov rdi, rax; jne 0x7c0d41; xor eax, eax; ret
16.	CVE-2022-1015	amd64	RoP	cli; ret pop rbp; ret mov rdim rax; ret pop rsi; ret pop rdi; ret add rsp, 0x90; pop rbx; pop rbp; ret

ПРИЛОЖЕНИЕ 3. СТАТИСТИКА ИСПОЛЬЗУЕМЫХ РЕГИСТРОВ И ИНСТРУКЦИЙ

Регистр	Использующие регистр модули	
	Количество	Доля, %
rax, rbx, rcx,rdx,rsp, rbp,rdi,rsi, r8-r15	3434	100
mmx0-7	0	0
st(0)	82	0,02387886
st(1)	85	0,02475248
st(2)	49	0,01426907
st(3)	40	0,01164822
st(4)	24	0,00698893
st(5)	13	0,00378567
st(6)	14	0,00407688
st(7)	9	0,00262085
xmm0	2017	0,58736168
xmm1	1857	0,54076878
xmm2	1692	0,49271986
xmm3	1626	0,47350029
xmm4	1505	0,43826441
xmm5	1439	0,41904485
xmm6	1403	0,40856144
xmm7	1372	0,39953407
xmm8	353	0,10279557
xmm9	330	0,09609785
xmm10	289	0,08415842
xmm11	267	0,07775189
xmm12	243	0,07076296
xmm13	218	0,06348282
xmm14	204	0,05940594
xmm15	189	0,05503786

Код оп.	Мнемоника	Число	Доля, %
89	mov	36017775	17,3345478
8b	mov	26010395	12,5182201
e8	call	16246377	7,8190171
8d	lea	12162756	5,8536618
85	test	6733742	3,2407991
0f1f	nop	6688158	3,2188605
e9	jmp	6361592	3,0616916
74	je	5774801	2,7792823
31	xor	5074753	2,4423649
0f84	je	4573281	2,2010176
83/0	add	3971048	1,9111764
c7/0	mov	3619708	1,7420843
39	cmp	3397874	1,6353206
83/7	cmp	3314482	1,5951859
75	jne	2888916	1,3903705
5d	pop	2393848	1,1521054
c3	ret	2392161	1,1512935
0f85	jne	2364427	1,1379457
83/5	sub	2305863	1,1097602
eb	jmp	2117636	1,0191707
be	mov	1959759	0,9431881
55	push	1955273	0,9410291
5b	pop	1883560	0,9065152
ba	mov	1712014	0,8239540
0fb6	movzx	1665184	0,8014157

Код оп.	Мнемоника	Число	Доля, %
53	push	1468685	0,7068452
01	add	1461832	0,7035470
b8	mov	1421406	0,6840908
84	test	1292554	0,6220773
5c	pop	1134723	0,5461167
ff/2	call	1089443	0,5243245
c6/0	mov	1065728	0,5129110
80/7	cmp	1010537	0,4863488
63	movsxd	999611	0,4810904
90	nop	968201	0,4659734
29	sub	965471	0,4646596
ff/4	jmp	956479	0,4603319
83/4	and	952699	0,4585127
c1/4	shl	943876	0,4542664
54	push	911392	0,4386325
c1/5	shr	898834	0,4325887
bf	mov	821729	0,3954797
90	xchg	815231	0,3923524
b9	mov	771216	0,3711690
88	mov	711311	0,3423380
68	push	647820	0,3117812
5e	pop	639240	0,3076519
0f95	setne	604760	0,2910574
3b	cmp	601648	0,2895597
b6	mov	598857	0,2882165
56	push	539037	0,2594264
09	or	534322	0,2571572
0fb7	movzx	504900	0,2429971

Код оп.	Мнемоника	Число	Доля, %
0f10	movsd	478857	0,2304631
f6/0	test	469256	0,2258424
5f	pop	464615	0,2236088
0f6f	movdqa	431496	0,2076694
03	add	426421	0,2052269
57	push	404879	0,1948592
0f11	movsd	352420	0,1696118
c1/7	sar	345590	0,1663247
76	jbe	324869	0,1563522
0f86	jbe	317454	0,1527835
0f87	ja	306230	0,1473816
81/7	cmp	306216	0,1473749
6a	push	290065	0,1396018
21	and	289248	0,1392086
0fef	pxor	288293	0,1387490
0f8e	jle	261481	0,1258449
77	ja	245145	0,1179828
0f10	movss	242813	0,1168605
0f28	movapd	239666	0,1153459
81/0	add	235234	0,1132129
7e	jle	232755	0,1120198
3c	cmp	226929	0,1092158
0f29	movaps	214632	0,1032976
0f44	cmove	210218	0,1011732
83/1	or	199211	0,0958758
0f11	movss	198482	0,0955249
0faf	imul	192912	0,0928442
81/4	and	190274	0,0915746

Код оп.	Мнемоника	Число	Доля, %
50	push	189007	0,0909648
0f94	sete	183628	0,0883760
81/5	sub	178801	0,0860529
0f59	mulsd	177488	0,0854210
0f28	movaps	173078	0,0832986
ff/6	push	171522	0,0825497
3d	cmp	169070	0,0813696
0ffe	paddd	167387	0,0805596
0f83	jae	162469	0,0781927
bd	mov	154959	0,0745783
0f58	addsd	154940	0,0745691
7f	jg	153705	0,0739748
0f8f	jg	150426	0,0723967
0f88	js	146758	0,0706313
a8	test	145561	0,0700552
2b	sub	144631	0,0696077
25	and	141994	0,0683385
0ff5	pmaddwd	139224	0,0670054
0f70	pshufd	132873	0,0639488
bb	mov	132307	0,0636764
0f2e	ucomisd	124631	0,0599821
0f5c	subsd	124196	0,0597728
b7	mov	122816	0,0591086
0f45	cmovne	120994	0,0582317
73	jae	118906	0,0572268
78	js	117289	0,0564486
0fbe	movsx	116022	0,0558388
0f7e	movq	115958	0,0558080

Код оп.	Мнемоника	Число	Доля, %
98	cdqe	112423	0,0541067
72	jb	111968	0,0538877
0f82	jb	110034	0,0529569
5a	pop	106723	0,0513634
0f59	mulss	105855	0,0509456
0f2a	cvtsi2sd	105286	0,0506718
d3/4	shl	97699	0,0470203
0fbf	movsx	96662	0,0465213
f7/2	not	91306	0,0439435
0f58	addss	88948	0,0428087
d1/5	shr	86160	0,0414669
58	pop	83045	0,0399677
db	fld	81807	0,0393719
d1/7	sar	75148	0,0361670
59	pop	73249	0,0352531
bc	mov	73155	0,0352079
f7/0	test	65536	0,0315410
0f5c	subss	64515	0,0310496
7c	jl	63228	0,0304302
0f16	movhps	61573	0,0296337
80/1	or	60198	0,0289719
f7/3	neg	59779	0,0287703
0f8d	jge	58924	0,0283588
0f6f	movdqu	56713	0,0272947
52	push	55873	0,0268904
7d	jge	55441	0,0266825
c1/0	rol	55088	0,0265126
0f2c	cvttsd2si	54584	0,0262701

Код оп.	Мнемоника	Число	Доля, %
80/4	and	54496	0,0262277
0f8c	jl	53033	0,0255236
c0/5	shr	52655	0,0253417
05	add	51503	0,0247872
0ffd	paddw	49642	0,0238916
0f72	psrad	48297	0,0232443
0f60	punpcklbw	47535	0,0228775
c5	vmovd	47273	0,0227514
69	imul	47125	0,0226802
33	xor	46366	0,0223149
0f2e	ucomiss	46294	0,0222803
38	cmp	45754	0,0220204
a6	cmps	45588	0,0219405
0ff9	psubw	43775	0,0210679
51	push	43007	0,0206983
0f5e	divsd	42584	0,0204947
c1/1	ror	42087	0,0202555
79	jns	41878	0,0201549
0f6b	packssdw	41211	0,0198339
0f7f	movdqu	41026	0,0197449
19	sbb	40183	0,0193392
db	fstp	39828	0,0191683
c5	vpaddd	38026	0,0183011
0f2a	cvtsi2ss	37777	0,0181812
0fa3	bt	37650	0,0181201
83/6	xor	37181	0,0178944
d9	fld	36822	0,0177216
f7/6	div	36775	0,0176990

Код оп.	Мнемоника	Число	Доля, %
d3/5	shr	36675	0,0176509
c9	leave	36635	0,0176316
0f6c	punpcklqdq	35044	0,0168659
a9	test	34868	0,0167812
0f5a	cvtss2sd	33290	0,0160217
0f46	cmovbe	32935	0,0158509
d9	fxch	32836	0,0158032
0f42	cmovb	32319	0,0155544
0f0b	ud2	31932	0,0153682
0f61	punpcklwd	30909	0,0148758
0f89	jns	30787	0,0148171
c5	vmovdqu	29715	0,0143012
23	and	29269	0,0140865
0f48	cmovs	29011	0,0139623
0f47	cmova	28728	0,0138261
0f4f	cmovg	28377	0,0136572
ab	stos	27485	0,0132279
0f4e	cmovle	27342	0,0131591
de	faddp	27225	0,0131028
0f97	seta	25411	0,0122297
0f58	addps	25360	0,0122052
7a	jp	25360	0,0122052
81/1	or	25192	0,0121243
0f4c	cmovl	25157	0,0121075
d3/7	sar	24610	0,0118442
0f6f	movq	24028	0,0115641
08	or	23933	0,0115184
0f59	mulps	23930	0,0115170

Код оп.	Мнемоника	Число	Доля, %
f7/7	idiv	23665	0,0113894
0f92	setb	22331	0,0107474
0f5c	subps	22114	0,0106430
c4	vpmaddwd	22042	0,0106083
f7/4	mul	21884	0,0105323
0f72	psrld	21532	0,0103629
0f58	addpd	21370	0,0102849
de	fmulp	21221	0,0102132
0f5c	subpd	20874	0,0100462
0f5a	cvtsd2ss	20607	0,0099177
c4	vmovd	20572	0,0099008
0f59	mulpd	19475	0,0093729
0fd6	movq	19048	0,0091674
0f6e	movd	18963	0,0091265
0f57	xorpd	18951	0,0091207
0fee	pmaxsw	18225	0,0087713
99	cdq	18171	0,0087453
c5	vmovapd	18040	0,0086822
0f43	cmovae	17930	0,0086293
0fdb	pand	17595	0,0084681
f7/5	imul	17520	0,0084320
0f49	cmovns	17429	0,0083882
6b	imul	17324	0,0083377
c5	vmovaps	17205	0,0082804
c4	vmovdqu	16873	0,0081206
0f62	punpckldq	16608	0,0079931
0f4d	cmovge	16534	0,0079574
c5	vmulps	16351	0,0078694

Код оп.	Мнемоника	Число	Доля, %
d8	fmul	16246	0,0078188
0b	or	16244	0,0078179
0f2c	cvttss2si	16216	0,0078044
0f96	setbe	15985	0,0076932
c5	vmulpd	15601	0,0075084
c5	vsubps	15537	0,0074776
0f7e	movd	15475	0,0074478
c5	vmovdqa	15459	0,0074401
c5	vsubpd	15436	0,0074290
0f69	punpckhwd	15391	0,0074073
0fc8	bswap	15122	0,0072779
0f9f	setg	15019	0,0072283
c5	vaddps	15007	0,0072225
0feb	por	14989	0,0072139
0f67	packuswb	14381	0,0069213
c5	vaddpd	14363	0,0069126
0f54	andpd	14286	0,0068755
de	fsubrp	13862	0,0066715
0f12	movhlps	13850	0,0066657
a5	movs	13649	0,0065690
0f5e	divss	13123	0,0063158
0fc6	shufps	12810	0,0061652
0f71	psllw	12786	0,0061536
c5	vpxor	12785	0,0061531
0f70	pshuflw	12609	0,0060684
ff/1	dec	12565	0,0060473
0f11	movups	12525	0,0060280
0f8a	jp	12515	0,0060232

Код оп.	Мнемоника	Число	Доля, %
0ff6	psadbw	12482	0,0060073
de	fsubp	12399	0,0059674
c5	vxorps	12314	0,0059265
c5	vpmaddwd	12308	0,0059236
c4	vpaddd	12276	0,0059082
d8	fadd	11806	0,0056820
83/3	sbb	11733	0,0056468
8a	mov	11607	0,0055862
0f3a	palignr	11302	0,0054394
0f7f	movdqa	11242	0,0054105
20	and	11133	0,0053581
c4	vinserti128	11028	0,0053075
c4	vmovlps	11022	0,0053046
2d	sub	11013	0,0053003
0f38	pmaddubsw	11010	0,0052989
0f71	psraw	10891	0,0052416
3a	cmp	10637	0,0051193
0fea	pminsw	10622	0,0051121
0f17	movhps	10475	0,0050414
c4	vpshufb	10332	0,0049726
0f57	xorps	10293	0,0049538
d8	fsub	10253	0,0049345
d0/5	shr	9887	0,0047584
c5	vpunpcklwd	9704	0,0046703
0f68	punpckhbw	9336	0,0044932
c5	vpaddw	9206	0,0044306
32	xor	9202	0,0044287
0f70	pshufhw	8994	0,0043286

Код оп.	Мнемоника	Число	Доля, %
c5	vmovups	8737	0,0042049
0f72	pslld	8600	0,0041390
c5	vpsubw	8371	0,0040288
0f93	setae	8354	0,0040206
c5	vpsrad	8238	0,0039648
0f38	pmulhrsw	8217	0,0039547
c5	vpunpckhwd	8175	0,0039344
ff/0	inc	7901	0,0038026
0f9e	setle	7883	0,0037939
99	cqo	7872	0,0037886
c4	vpermilpd	7844	0,0037751
c4	vpmaddubsw	7821	0,0037641
c4	vextractf128	7534	0,0036259
0ffa	psubd	7460	0,0035903
c5	vpsadbw	7337	0,0035311
0ff4	pmuludq	7297	0,0035119
c5	vpsrld	7110	0,0034219
c5	vaddsd	7060	0,0033978
0d	or	6958	0,0033487
0f73	psrldq	6955	0,0033473
30	xor	6936	0,0033381
0fbd	bsr	6816	0,0032804
0f7f	movq	6758	0,0032525
0fca	bswap	6726	0,0032371
c4	vextracti128	6692	0,0032207
0f9c	setl	6650	0,0032005
c5	vmovupd	6560	0,0031572
d1/0	rol	6554	0,0031543

Код оп.	Мнемоника	Число	Доля, %
0f73	psrlq	6258	0,0030118
83/2	adc	6244	0,0030051
0fd5	pmullw	6235	0,0030008
c5	vpand	6206	0,0029868
c4	vpxor	5971	0,0028737
c4	vinsertf128	5824	0,0028030
0f38	pshufb	5818	0,0028001
0f5f	maxsd	5817	0,0027996
0f74	pcmpeqb	5806	0,0027943
c4	vpermilps	5803	0,0027929
0fc6	shufpd	5609	0,0026995
dd	fstp	5586	0,0026884
0f38	phaddd	5453	0,0026244
0f5d	minsd	5434	0,0026153
c4	vsubps	5361	0,0025801
b0	mov	5344	0,0025719
c5	vmovq	5328	0,0025642
c4	vaddps	5312	0,0025565
c4	vsubpd	5268	0,0025354
c4	vaddpd	5243	0,0025233
c5	vmovlps	5243	0,0025233
0f6a	punpckhdq	4951	0,0023828
0f9b	setnp	4950	0,0023823
c4	vbroadcasti128	4665	0,0022452
0f54	andps	4536	0,0021831
c5	vpackssdw	4442	0,0021378
0f13	movlps	4379	0,0021075
0f9d	setge	4311	0,0020748

Код оп.	Мнемоника	Число	Доля, %
c4	vpermq	4307	0,0020729
c5	vmovhps	4212	0,0020271
c5	vmovlhps	4197	0,0020199
c5	vmovsd	4188	0,0020156
0a	or	4182	0,0020127
c4	vmovhps	4143	0,0019939
dd	fst	4122	0,0019838
c5	vxorpd	4063	0,0019554
ae	scas	4057	0,0019525
11	adc	4029	0,0019391
c4	vaddsd	4008	0,0019290
0fd4	paddq	3972	0,0019116
0fd7	pmovmskb	3909	0,0018813
00	add	3898	0,0018760
c4	vpsrld	3871	0,0018630
0fdf	pandn	3825	0,0018409
c4	vpmulhrsw	3812	0,0018346
c4	vmovapd	3762	0,0018106
c4	vpabsw	3712	0,0017865
c5	vzeroupper	3683	0,0017725
0f10	movups	3637	0,0017504
0f71	psrlw	3602	0,0017336
c5	vsubsd	3559	0,0017129
22	and	3538	0,0017028
80/0	add	3534	0,0017008
0fd8	psubusb	3526	0,0016970
98	cwde	3487	0,0016782
c4	vmulps	3485	0,0016773

Код оп.	Мнемоника	Число	Доля, %
0fc9	bswap	3332	0,0016036
0f64	pcmpgtb	3302	0,0015892
c4	vmulpd	3243	0,0015608
02	add	3209	0,0015444
0fdd	paddusw	3118	0,0015006
0fce	bswap	3081	0,0014828
c4	vpsrad	3036	0,0014612
0f12	movlps	3035	0,0014607
0f73	pslldq	3021	0,0014539
c5	vmulsd	2917	0,0014039
c4	vmovupd	2912	0,0014015
0f2f	comisd	2852	0,0013726
c4	vmovups	2850	0,0013716
0f51	sqrtsd	2816	0,0013553
c5	vpslld	2810	0,0013524
81/6	xor	2793	0,0013442
c4	rorx	2762	0,0013293
c5	vpor	2760	0,0013283
0f6d	punpckhqdq	2703	0,0013009
c4	vmovdqa	2700	0,0012994
c4	vpalignr	2689	0,0012942
0f66	pcmpgtd	2665	0,0012826
dc	fmul	2637	0,0012691
c4	vxorps	2633	0,0012672
c5	vpmaxsw	2624	0,0012629
0f73	psllq	2569	0,0012364
0f14	unpcklpd	2562	0,0012330
0f14	unpcklps	2539	0,0012220

Код оп.	Мнемоника	Число	Доля, %
0f5d	minss	2512	0,0012090
0f5f	maxss	2494	0,0012003
c4	vpunpcklwd	2481	0,0011940
0fae	mfence	2455	0,0011815
c4	vpunpckhwd	2453	0,0011806
0fe0	pavgb	2402	0,0011560
c4	vpaddw	2392	0,0011512
c5	vpackuswb	2369	0,0011401
c5	vpunpcklbw	2359	0,0011353
a4	movs	2333	0,0011228
0f15	unpckhpd	2303	0,0011084
0f38	pabsw	2289	0,0011016
0f15	unpckhps	2285	0,0010997
d1/1	ror	2272	0,0010935
0f38	packusdw	2237	0,0010766
0fac	shrd	2189	0,0010535
c4	vphaddd	2178	0,0010482
0f56	orpd	2173	0,0010458
0f9a	setp	2148	0,0010338
c5	vpminsw	2144	0,0010319
c5	vunpcklpd	2142	0,0010309
f4	hlt	2138	0,0010290
0ff8	psubb	2120	0,0010203
80/5	sub	2092	0,0010068
c5	vpandn	2049	0,0009861
c4	vmovaps	2039	0,0009813
c5	vmovshdup	2017	0,0009707
c5	vmovsldup	2013	0,0009688

Код оп.	Мнемоника	Число	Доля, %
0f38	aesenc	1984	0,0009549
0fed	paddsw	1940	0,0009337
c5	vmovhlps	1939	0,0009332
0fba	bt	1912	0,0009202
0ff0	lddqu	1882	0,0009058
24	and	1875	0,0009024
c4	vsubsd	1872	0,0009010
86	xchg	1871	0,0009005
0fe5	pmulhw	1801	0,0008668
0f55	andnpd	1789	0,0008610
0fa4	shld	1772	0,0008528
c4	vmovq	1772	0,0008528
c4	vmulsd	1759	0,0008466
0f50	movmskpd	1735	0,0008350
0f05	syscall	1700	0,0008182
0fc4	pinsrw	1673	0,0008052
35	xor	1666	0,0008018
0fbc	tzcnt	1661	0,0007994
c4	vpsubw	1644	0,0007912
d3/0	rol	1636	0,0007874
0fcb	bswap	1634	0,0007864
c4	vpackusdw	1616	0,0007777
c5	vpaddq	1600	0,0007700
0fdc	paddusb	1595	0,0007676
c4	vfmaddsd	1566	0,0007537
c4	vpor	1566	0,0007537
c4	vpslld	1534	0,0007383
c5	vpunpckldq	1529	0,0007359

Код оп.	Мнемоника	Число	Доля, %
c5	vandpd	1520	0,0007315
c4	vpackssdw	1499	0,0007214
c5	vpunpcklqdq	1490	0,0007171
0c	or	1488	0,0007161
c5	vpshuflw	1463	0,0007041
0fcf	bswap	1451	0,0006983
c5	vpunpckhbw	1447	0,0006964
87	xchg	1443	0,0006945
70	jo	1435	0,0006906
0f65	pcmpgtw	1420	0,0006834
0fe4	pmulhuw	1387	0,0006675
0fcd	bswap	1368	0,0006584
fc	cld	1367	0,0006579
0f56	orps	1360	0,0006545
2c	sub	1357	0,0006531
c5	vshufps	1335	0,0006425
c5	vpunpckhdq	1318	0,0006343
0f76	pcmpeqd	1314	0,0006324
0ffc	paddb	1313	0,0006319
0f16	movlhps	1306	0,0006285
0fda	pminub	1298	0,0006247
d1/4	shl	1289	0,0006204
0f70	pshufw	1286	0,0006189
c5	vpunpckhqdq	1280	0,0006160
c4	vmovlhps	1278	0,0006151
0f38	pmovzxbw	1275	0,0006136
c4	vaesenclast	1266	0,0006093
c5	vucomisd	1260	0,0006064

Код оп.	Мнемоника	Число	Доля, %
c5	vpmuludq	1255	0,0006040
2a	sub	1246	0,0005997
c4	vpand	1238	0,0005958
c5	vpsubd	1220	0,0005872
0f38	aesdec	1197	0,0005761
c4	vpbroadcastd	1186	0,0005708
0f51	sqrtss	1143	0,0005501
0fe3	pavgw	1141	0,0005491
0f2f	comiss	1129	0,0005434
80/6	xor	1102	0,0005304
db	fucomi	1100	0,0005294
ac	lods	1098	0,0005284
0f75	pcmpeqw	1078	0,0005188
c5	vaddsubpd	1075	0,0005174
c4	vpsadbw	1064	0,0005121
0f3a	pinsrb	1062	0,0005111
c5	vaddsubps	1033	0,0004972
d8	fsubr	1028	0,0004948
c5	vpsrldq	1023	0,0004923
c5	vpcmpgtb	988	0,0004755
d9	fldz	960	0,0004620
dc	fsub	959	0,0004615
c4	vpsrldq	943	0,0004538
c4	vpaddq	912	0,0004389
13	adc	888	0,0004274
c4	vaesenc	882	0,0004245
0f55	andnps	875	0,0004211
c5	vpshufd	875	0,0004211

Код оп.	Мнемоника	Число	Доля, %
df	fucomip	871	0,0004192
c4	vpmovzxbw	866	0,0004168
c5	vpsrlq	863	0,0004153
c5	vpsllw	855	0,0004115
aa	stos	815	0,0003922
b5	mov	805	0,0003874
0f5b	cvtdq2ps	782	0,0003764
d3/1	ror	777	0,0003740
0fcc	bswap	753	0,0003624
0f63	packsswb	752	0,0003619
0fe2	psrad	740	0,0003561
c4	vpunpckhqdq	717	0,0003451
c4	vpunpcklqdq	716	0,0003446
b4	mov	709	0,0003412
0fc2	cmpltsd	706	0,0003398
b2	mov	702	0,0003379
c4	vpextrd	697	0,0003355
7b	jnp	692	0,0003330
c4	vshufps	688	0,0003311
0f12	movddup	662	0,0003186
c5	vpsraw	658	0,0003167
71	jno	653	0,0003143
df	fild	638	0,0003071
1b	sbb	631	0,0003037
0f38	pabsd	624	0,0003003
34	xor	623	0,0002998
c0/7	sar	617	0,0002969
d9	fldcw	615	0,0002960

Код оп.	Мнемоника	Число	Доля, %
0f38	phaddw	613	0,0002950
c4	vpsubd	613	0,0002950
dc	fadd	613	0,0002950
0f13	movlpd	612	0,0002945
0f77	emms	612	0,0002945
0f8b	jnp	612	0,0002945
c4	vpunpckldq	610	0,0002936
0fbc	bsf	609	0,0002931
c4	vxorpd	607	0,0002921
0f3a	pextrd	604	0,0002907
0f6e	movq	601	0,0002892
dc	fsubr	595	0,0002864
c4	vperm2i128	579	0,0002787
c4	vpunpckhdq	578	0,0002782
c5	vpavgw	577	0,0002777
dd	fld	574	0,0002763
0fec	paddsb	573	0,0002758
c5	vpcmpeqw	572	0,0002753
b1	mov	562	0,0002705
d9	fchs	559	0,0002690
0f38	pmulld	558	0,0002686
f6/4	mul	557	0,0002681
c4	vpinsrd	550	0,0002647
c4	vbroadcastss	541	0,0002604
0f38	adcx	533	0,0002565
0f17	movhpd	526	0,0002532
04	add	523	0,0002517
0fc2	cmplesd	521	0,0002507

Код оп.	Мнемоника	Число	Доля, %
0f5b	cvtps2dq	520	0,0002503
c4	vpbroadcastq	515	0,0002479
0fb1	cmpxchg	514	0,0002474
c4	vpunpcklbw	514	0,0002474
c4	vpandn	510	0,0002455
0f0f	pavgusb	507	0,0002440
f6/2	not	503	0,0002421
c4	vucomisd	496	0,0002387
c5	vpaddusw	491	0,0002363
0fe9	psubsw	482	0,0002320
0fa2	cpuid	479	0,0002305
d9	fabs	475	0,0002286
0f18	prefetcht0	474	0,0002281
0fde	pmaxub	472	0,0002272
c4	mulx	472	0,0002272
c4	vpermd	470	0,0002262
c5	vpsubusb	469	0,0002257
28	sub	468	0,0002252
0f38	ptest	459	0,0002209
10	adc	439	0,0002113
0f10	movupd	434	0,0002089
c4	vpbroadcastw	432	0,0002079
c5	kmovw	432	0,0002079
b3	mov	429	0,0002065
c4	vunpcklpd	428	0,0002060
c5	vunpcklps	427	0,0002055
c5	vunpckhps	426	0,0002050
0f38	adox	424	0,0002041

Код оп.	Мнемоника	Число	Доля, %
c4	vpmuludq	422	0,0002031
c5	vpsrlw	416	0,0002002
c4	andn	414	0,0001992
c4	vpabsd	414	0,0001992
d9	fld1	402	0,0001935
c4	vfmsubsd	396	0,0001906
0fc2	cmpnltsd	394	0,0001896
92	xchg	385	0,0001853
c4	vpunpckhbw	383	0,0001843
15	adc	379	0,0001824
0fc5	pextrw	375	0,0001805
c5	vpaddb	372	0,0001790
0f3a	pcmpistri	371	0,0001786
c5	vcvtdq2ps	371	0,0001786
0fe7	movntdq	369	0,0001776
1c	sbb	365	0,0001757
14	adc	364	0,0001752
c4	vperm2f128	364	0,0001752
0f18	prefetchnta	362	0,0001742
df	fnstsw	360	0,0001733
c4	vpmulld	359	0,0001728
c2	ret	356	0,0001713
c5	vpavgb	356	0,0001713
97	xchg	355	0,0001709
9b	fwait	353	0,0001699
de	fdivrp	352	0,0001694
91	xchg	351	0,0001689
a7	cmps	350	0,0001684

Код оп.	Мнемоника	Число	Доля, %
c5	vpcmpeqb	347	0,0001670
9d	popf	343	0,0001651
8f/5	vphadduwq	341	0,0001641
c5	vpslldq	341	0,0001641
d7	xlat	341	0,0001641
18	sbb	337	0,0001622
0fe7	movntq	336	0,0001617
d0/7	sar	335	0,0001612
d9	fxam	335	0,0001612
f8	clc	335	0,0001612
f9	stc	333	0,0001603
0fd9	psubusw	332	0,0001598
ad	lods	331	0,0001593
fd	std	329	0,0001583
94	xchg	328	0,0001579
93	xchg	326	0,0001569
9e	sahf	326	0,0001569
e3	jrcxz	326	0,0001569
c4	vpsrlq	325	0,0001564
d8	fdiv	324	0,0001559
c4	vpmaxsw	318	0,0001530
0f3a	pextrb	317	0,0001526
96	xchg	317	0,0001526
c4	vpsraw	314	0,0001511
8c	mov	312	0,0001502
8e	mov	312	0,0001502
1d	sbb	311	0,0001497
0f3a	pextrw	308	0,0001482

Код оп.	Мнемоника	Число	Доля, %
c4	vpmovsxwd	308	0,0001482
9c	pushf	306	0,0001473
c5	vpsllq	306	0,0001473
df	fistp	306	0,0001473
1a	sbb	304	0,0001463
0f38	aesenclast	298	0,0001434
0fe1	psraw	297	0,0001429
af	scas	297	0,0001429
c4	vpsubusb	294	0,0001415
f5	cmc	294	0,0001415
d9	fnstcw	287	0,0001381
c4	vfnmaddsd	280	0,0001348
95	xchg	274	0,0001319
0fae	stmxcsr	273	0,0001314
9f	lahf	273	0,0001314
c5	vmovddup	272	0,0001309
c5	vpsubb	269	0,0001295
c4	vmovshdup	266	0,0001280
c4	vmovsldup	266	0,0001280
0fe6	cvtdq2pd	260	0,0001251
db	fild	260	0,0001251
c4	vpackuswb	259	0,0001247
0f2d	cvtss2si	256	0,0001232
0f38	sha256rnds2	256	0,0001232
98	cbw	256	0,0001232
0ff1	psllw	255	0,0001227
0fae	ldmxcsr	253	0,0001218
c4	vpminsw	251	0,0001208

Код оп.	Мнемоника	Число	Доля, %
12	adc	249	0,0001198
c4	vfmaddps	248	0,0001194
0fe8	psubsb	246	0,0001184
c4	vmovsd	235	0,0001131
0fc2	cmpnlesd	234	0,0001126
c0/4	shl	234	0,0001126
c4	vpmovzxdq	231	0,0001112
0fc2	cmpltss	229	0,0001102
8f/5	vpperm	228	0,0001097
0f5d	minps	224	0,0001078
c4	vaesdec	224	0,0001078
c4	vaddsubps	220	0,0001059
de	fdivp	218	0,0001049
0f5e	divps	213	0,0001025
0fc2	cmpnless	212	0,0001020
0f5f	maxps	209	0,0001006
0f38	pmovsxwd	205	0,0000987
c5	vpacksswb	204	0,0000982
0f38	psignw	203	0,0000977
0f5a	cvtps2pd	202	0,0000972
c4	vmovss	201	0,0000967
c4	vunpckhps	199	0,0000958
c4	vunpcklps	199	0,0000958
fe/0	inc	196	0,0000943
0f80	jo	195	0,0000938
c5	vpmullw	195	0,0000938
fe/1	dec	195	0,0000938
0f12	movsldup	194	0,0000934

Код оп.	Мнемоника	Число	Доля, %
c4	vpclmullqlqdq	192	0,0000924
c4	vpcmpistri	192	0,0000924
c4	vmovhlps	190	0,0000914
0f7c	haddps	187	0,0000900
c5	vpcmpgtw	185	0,0000890
c4	vpshufd	184	0,0000886
0f3a	pclmullqlqdq	183	0,0000881
c5	vdivsd	183	0,0000881
c4	vpextrw	182	0,0000876
0fc2	cmpnltss	179	0,0000861
c5	vpaddusb	179	0,0000861
0fc2	cmpless	178	0,0000857
c4	vaddsubpd	174	0,0000837
c4	vpsubq	174	0,0000837
0f5a	cvtpd2ps	172	0,0000828
c4	vpmovzxwd	172	0,0000828
c4	vpcmpgtb	171	0,0000823
c4	vpblendw	166	0,0000799
c5	vunpckhpd	166	0,0000799
0f38	aesdeclast	164	0,0000789
0f3a	aeskeygenassist	163	0,0000784
0f38	sha1nexte	160	0,0000770
0f3a	mpsadbw	160	0,0000770
0f3a	sha1rnds4	160	0,0000770
f6/6	div	160	0,0000770
dc	fdiv	157	0,0000756
0fe6	cvttpd2dq	156	0,0000751
c4	vpblendd	155	0,0000746

Код оп.	Мнемоника	Число	Доля, %
c0/0	rol	151	0,0000727
0fc2	cmpltps	149	0,0000717
c4	vpshuflw	147	0,0000707
c4	vpcmpgtw	144	0,0000693
c5	vcvtps2dq	143	0,0000688
0f51	sqrtps	138	0,0000664
0f3a	pblendw	137	0,0000659
0fae	sfence	135	0,0000650
c4	vpmuldq	135	0,0000650
c5	vpaddsw	133	0,0000640
c4	vpsrlw	132	0,0000635
db	fistp	131	0,0000630
c4	vunpckhpd	130	0,0000626
0f18	prefetcht1	128	0,0000616
0f38	sha1msg1	128	0,0000616
0f38	sha1msg2	128	0,0000616
0ff2	pslld	126	0,0000606
c4	vpclmulhqhqdq	126	0,0000606
c5	vpinsrw	125	0,0000602
0f3a	pinsrd	121	0,0000582
0f12	movlpd	119	0,0000573
c4	vfmadd213pd	119	0,0000573
0f16	movhpd	113	0,0000544
c4	vpsllq	113	0,0000544
0f4a	cmovp	112	0,0000539
c4	vpaddb	112	0,0000539
0f2d	cvtsd2si	111	0,0000534
0f3a	pclmulhqhqdq	109	0,0000525

Код оп.	Мнемоника	Число	Доля, %
c0/1	ror	109	0,0000525
0fd0	addsubps	108	0,0000520
0fd6	movq2dq	107	0,0000515
8f/5	vphadddq	107	0,0000515
c4	vpabsb	107	0,0000515
c4	vmovddup	106	0,0000510
c4	vpcmpeqq	105	0,0000505
c5	vpmovmskb	103	0,0000496
c4	vpmovsxbw	102	0,0000491
c5	vorpd	102	0,0000491
c4	vpslldq	99	0,0000476
dc	fdivr	99	0,0000476
0f0e	femms	98	0,0000472
d8	fdivr	98	0,0000472
df	fist	97	0,0000467
0f38	sha256msg1	96	0,0000462
0f38	sha256msg2	96	0,0000462
8f/5	vphaddwq	95	0,0000457
0f31	rdtsc	93	0,0000448
c4	vpsignw	93	0,0000448
c5	vpcmpeqd	93	0,0000448
8f/0	pop	91	0,0000438
c4	vphaddw	91	0,0000438
d9	fsqrt	91	0,0000438
c4	vpclmullqhqdq	90	0,0000433
c4	vbroadcastf128	89	0,0000428
c5	vcvtdq2pd	88	0,0000424
81/3	sbb	85	0,0000409

Код оп.	Мнемоника	Число	Доля, %
d0/4	shl	85	0,0000409
d8	fcom	85	0,0000409
0f16	movshdup	83	0,0000399
0fd6	movdq2q	83	0,0000399
db	fist	81	0,0000390
f6/3	neg	81	0,0000390
0f38	pmovzxwd	80	0,0000385
8f/5	vprotd	80	0,0000385
c5	vpaddsb	80	0,0000385
d9	fnstenv	80	0,0000385
da	fcmove	76	0,0000366
8f/1	vphadduwq	75	0,0000361
c4	vpinsrb	74	0,0000356
d9	fldenv	74	0,0000356
0f5e	divpd	69	0,0000332
d1/3	rcr	69	0,0000332
d9	fstp	69	0,0000332
0fc2	cmpleps	68	0,0000327
c4	vpsubb	68	0,0000327
0f50	movmskps	67	0,0000322
c4	vpcmpeqw	66	0,0000318
c4	vpmullw	66	0,0000318
0f38	pmuldq	64	0,0000308
c5	vldmxcsr	64	0,0000308
c4	vpavgw	63	0,0000303
d0/0	rol	63	0,0000303
c5	vstmxcsr	62	0,0000298
dc	fcom	62	0,0000298

Код оп.	Мнемоника	Число	Доля, %
c4	vfmadd213ps	61	0,0000294
c4	vpavgb	61	0,0000294
d0/1	ror	60	0,0000289
da	fisub	60	0,0000289
0fc2	cmpunordsd	58	0,0000279
0f2a	cvtpi2ps	56	0,0000270
c5	vandnpd	56	0,0000270
0f38	movbe	54	0,0000260
c4	vpaddusb	54	0,0000260
c4	vpmaxsd	54	0,0000260
c4	vpminuw	54	0,0000260
de	fidiv	54	0,0000260
de	fisub	54	0,0000260
0f98	sets	52	0,0000250
0fc2	cmpnltps	52	0,0000250
80/2	adc	52	0,0000250
c5	vzeroall	51	0,0000245
d2/1	ror	51	0,0000245
f6/5	imul	51	0,0000245
0ffb	psubq	50	0,0000241
c5	vpminub	50	0,0000241
d8	fcomp	50	0,0000241
0fd3	psrlq	49	0,0000236
da	fidiv	49	0,0000236
de	ficom	49	0,0000236
0f38	pmovzxbd	48	0,0000231
0f3a	dpps	48	0,0000231
0f3a	pclmullqhqdq	48	0,0000231

Код оп.	Мнемоника	Число	Доля, %
8f/1	vprotq	48	0,0000231
c5	vcvttsd2si	48	0,0000231
d9	fst	47	0,0000226
0f38	pmovsxbw	46	0,0000221
dc	fcomp	46	0,0000221
c5	vpsubusw	45	0,0000217
d3/2	rcl	44	0,0000212
0ff3	psllq	43	0,0000207
c4	vblendps	43	0,0000207
d1/2	rcl	43	0,0000207
da	ficom	43	0,0000207
0fa5	shld	42	0,0000202
0fad	shrd	42	0,0000202
0fd1	psrlw	42	0,0000202
8f/5	vphadduwd	42	0,0000202
c4	vpsllw	42	0,0000202
de	fiadd	41	0,0000197
0f01	xgetbv	40	0,0000193
80/3	sbb	40	0,0000193
c5	kxnorw	40	0,0000193
c5	vcmpltsd	40	0,0000193
d2/0	rol	40	0,0000193
c1/2	rcl	39	0,0000188
d2/2	rcl	39	0,0000188
d2/7	sar	39	0,0000188
c5	vpmaxub	38	0,0000183
0f38	crc32	37	0,0000178
df	fcomip	37	0,0000178

Код оп.	Мнемоника	Число	Доля, %
81/2	adc	36	0,0000173
d0/3	rcr	36	0,0000173
d2/4	shl	36	0,0000173
c0/3	rcr	35	0,0000168
c1/3	rcr	35	0,0000168
c4	vpbroadcastb	35	0,0000168
c4	vpinsrq	35	0,0000168
c5	vandps	35	0,0000168
c5	vpcmpgtd	35	0,0000168
da	fiadd	35	0,0000168
0fc2	cmpunordss	34	0,0000164
d2/5	shr	34	0,0000164
0f38	phsubd	33	0,0000159
0f5b	cvttps2dq	33	0,0000159
dd	fnsave	33	0,0000159
0f38	pminsd	32	0,0000154
0fc2	cmpneqps	32	0,0000154
c4	vpmovmskb	32	0,0000154
d0/2	rcl	32	0,0000154
d9	fldln2	32	0,0000154
d9	fldpi	32	0,0000154
dd	fisttp	32	0,0000154
0fc2	cmpeqps	31	0,0000149
d2/3	rcr	31	0,0000149
c4	vandpd	30	0,0000144
c5	vcvttpd2dq	30	0,0000144
de	fisubr	30	0,0000144
f6/7	idiv	30	0,0000144

Код оп.	Мнемоника	Число	Доля, %
0f38	pmovsxbd	29	0,0000140
0f38	psignd	29	0,0000140
0fae	lfence	29	0,0000140
0fbd	lzcnt	29	0,0000140
c0/2	rcl	29	0,0000140
c4	vpaddsw	29	0,0000140
c5	vmovntdq	29	0,0000140
c5	vmovss	29	0,0000140
d3/3	rcr	29	0,0000140
da	fisubr	29	0,0000140
c4	vfmadd231ps	28	0,0000135
c5	vcmpnltps	28	0,0000135
da	fcmovu	28	0,0000135
db	fisttp	28	0,0000135
de	fidivr	28	0,0000135
df	fisttp	28	0,0000135
0f3a	pclmulhqlqdq	27	0,0000130
0fc2	cmpordsd	27	0,0000130
d9	frndint	27	0,0000130
de	ficomp	27	0,0000130
0f29	movapd	26	0,0000125
c4	vpshufhw	26	0,0000125
c4	vpsubusw	26	0,0000125
c5	vpmulhw	26	0,0000125
db	fcmovnb	26	0,0000125
dd	fnstsw	26	0,0000125
0f99	setns	25	0,0000120
0fc2	cmpordss	25	0,0000120

Код оп.	Мнемоника	Число	Доля, %
c4	vpaddsb	25	0,0000120
c4	vpminsd	25	0,0000120
d9	fyl2x	25	0,0000120
da	fimul	25	0,0000120
0f38	phsubw	24	0,0000116
c4	vdivsd	24	0,0000116
c4	vextractps	24	0,0000116
c4	vmovmskps	24	0,0000116
da	ficomp	24	0,0000116
dd	frstor	24	0,0000116
de	fimul	24	0,0000116
df	fbstp	24	0,0000116
07	pop	23	0,0000111
0f38	aesimc	23	0,0000111
d9	fscale	23	0,0000111
db	fcmovnbe	23	0,0000111
c4	vfnmadd231ps	22	0,0000106
0fba	bts	21	0,0000101
c4	shlx	21	0,0000101
c5	vmovmskps	21	0,0000101
0fd2	psrld	20	0,0000096
c4	shrx	20	0,0000096
c4	vmpsadbw	20	0,0000096
c5	vcvtsi2sd	19	0,0000091
da	fidivr	19	0,0000091
db	fcmovnu	19	0,0000091
df	fbld	19	0,0000091
df	ffreep	19	0,0000091

Код оп.	Мнемоника	Число	Доля, %
0f3a	pcmpestri	18	0,0000087
0fb8	popcnt	18	0,0000087
c4	vfnmadd231pd	18	0,0000087
c4	vpclmulhqlqdq	18	0,0000087
c4	vpextrq	18	0,0000087
c5	vdivpd	18	0,0000087
c5	vpsubq	18	0,0000087
d9	fprem	18	0,0000087
dd	fucom	18	0,0000087
c4	vpcmpeqd	17	0,0000082
d9	fyl2xp1	17	0,0000082
0f38	pmaxsd	16	0,0000077
0f4b	cmovnp	16	0,0000077
8f/5	vprotq	16	0,0000077
c4	vaesdeclast	16	0,0000077
c4	vcmpnltps	16	0,0000077
c4	vfmadd231pd	16	0,0000077
c4	vorpd	16	0,0000077
c5	vpsubsb	16	0,0000077
0f2b	movntps	15	0,0000072
0fab	bts	15	0,0000072
c4	pext	15	0,0000072
c4	vpblendvb	15	0,0000072
0ff7	maskmovdqu	14	0,0000067
c4	vbroadcastsd	14	0,0000067
c4	vpinsrw	14	0,0000067
c4	vpsignd	14	0,0000067
c5	vpshufhw	14	0,0000067

Код оп.	Мнемоника	Число	Доля, %
db	fcomi	13	0,0000063
0f3a	insertps	12	0,0000058
0f3a	roundps	12	0,0000058
0fc2	cmpnleps	12	0,0000058
82/4	and	12	0,0000058
8f/5	vpmadcswd	12	0,0000058
c4	vcvttsd2si	12	0,0000058
c4	vgatherqpd	12	0,0000058
c4	vpcmpgtd	12	0,0000058
c4	vpsignb	12	0,0000058
0f2d	cvtps2pi	11	0,0000053
c4	vptest	11	0,0000053
c5	vpmulhuw	11	0,0000053
d9	f2xm1	11	0,0000053
db	fneni(8087	11	0,0000053
0f3a	pinsrq	10	0,0000048
0fc2	cmpltpd	10	0,0000048
c4	vcvtsi2sd	10	0,0000048
c4	vpcmpeqb	10	0,0000048
c4	vpsubsb	10	0,0000048
c5	vcvtps2pd	10	0,0000048
c5	vpextrw	10	0,0000048
da	fcmovb	10	0,0000048
dd	fucomp	10	0,0000048
0f00	sldt	9	0,0000043
0f52	rsqrtps	9	0,0000043
d9	fldl2e	9	0,0000043
da	fucompp	9	0,0000043

Код оп.	Мнемоника	Число	Доля, %
0f01	xend	8	0,0000039
0f0f	pfmul	8	0,0000039
0f38	blendvps	8	0,0000039
0f3a	roundpd	8	0,0000039
0f3a	roundsd	8	0,0000039
0f3a	roundss	8	0,0000039
0f53	rcpps	8	0,0000039
0fc2	cmpnlepd	8	0,0000039
8f/1	vpperm	8	0,0000039
8f/5	vpmacsww	8	0,0000039
c4	vcmpltsd	8	0,0000039
c4	vcvtph2ps	8	0,0000039
c4	vcvtps2ph	8	0,0000039
c4	vfmsub213pd	8	0,0000039
c4	vgatherdpd	8	0,0000039
c4	vmovntdq	8	0,0000039
c4	vmovntps	8	0,0000039
c4	vpermps	8	0,0000039
c4	vpextrb	8	0,0000039
c4	vphminposuw	8	0,0000039
c4	vpmaxub	8	0,0000039
c5	vcmplesd	8	0,0000039
c5	vcmpnle_uqpd	8	0,0000039
c5	vcvtpd2ps	8	0,0000039
c5	vmovmskpd	8	0,0000039
0f38	psignb	7	0,0000034
c5	vhaddps	7	0,0000034
c5	vrcpps	7	0,0000034

Код оп.	Мнемоника	Число	Доля, %
d9	fpatan	7	0,0000034
dd	ffree	7	0,0000034
0f18	prefetcht2	6	0,0000029
0f38	pblendvb	6	0,0000029
0f38	pminud	6	0,0000029
0f38	pminuw	6	0,0000029
0f5d	minpd	6	0,0000029
0f5f	maxpd	6	0,0000029
0f90	seto	6	0,0000029
0fc7	rdrand	6	0,0000029
0ff7	maskmovq	6	0,0000029
c4	vcvtsi2ss	6	0,0000029
c4	vfmadd132ps	6	0,0000029
c4	vfnmadd132pd	6	0,0000029
c4	vfnmadd132ps	6	0,0000029
c4	vphsubw	6	0,0000029
c4	vroundpd	6	0,0000029
c5	vcmpgt_oqpd	6	0,0000029
c5	vcmpnle_uqps	6	0,0000029
d9	fldlg2	6	0,0000029
d9	fxtract	6	0,0000029
da	fcmovbe	6	0,0000029
db	fcmovne	6	0,0000029
db	fnclex	6	0,0000029
0f0f	pmulhrw	5	0,0000024
0f38	pabsb	5	0,0000024
0f52	rsqrtss	5	0,0000024
0fa7	xcrypt-ecb	5	0,0000024

Код оп.	Мнемоника	Число	Доля, %
0fba	btc	5	0,0000024
c4	les	5	0,0000024
c4	vfnmadd213ps	5	0,0000024
c4	vhaddps	5	0,0000024
c5	vdivss	5	0,0000024
c5	vmovntps	5	0,0000024
0f06	clts	4	0,0000019
0f08	invd	4	0,0000019
0f0d	prefetch	4	0,0000019
0f0f	pf2id	4	0,0000019
0f2a	cvtpi2pd	4	0,0000019
0f38	phminposuw	4	0,0000019
0f3a	blendps	4	0,0000019
0f78	vmread	4	0,0000019
0fae	clflush	4	0,0000019
0fae	xrstor	4	0,0000019
0fae	xrstor64	4	0,0000019
0fb5	lgs	4	0,0000019
0fba	btr	4	0,0000019
87	xacquire	4	0,0000019
8f/5	vphaddubq	4	0,0000019
8f/5	vpmacsdql	4	0,0000019
c4	bzhi	4	0,0000019
c4	vcvtps2pd	4	0,0000019
c4	vfmadd132pd	4	0,0000019
c4	vfnmadd213pd	4	0,0000019
c5	vaddss	4	0,0000019
c5	vandnps	4	0,0000019

Код оп.	Мнемоника	Число	Доля, %
c5	vcmpgtsd	4	0,0000019
c5	vdivps	4	0,0000019
c5	vmaxsd	4	0,0000019
c5	vminps	4	0,0000019
c5	vminsd	4	0,0000019
de	fcompp	4	0,0000019
0f00	str	3	0,0000014
0f0f	pswapd	3	0,0000014
0f81	jno	3	0,0000014
0fa7	xcrypt-ctr	3	0,0000014
0fc1	xadd	3	0,0000014
0fc7	rdseed	3	0,0000014
c5	vcvttps2dq	3	0,0000014
c5	vmulss	3	0,0000014
0f01	sgdt	2	0,0000010
0f01	xtest	2	0,0000010
0f21	mov	2	0,0000010
0f23	mov	2	0,0000010
0f30	wrmsr	2	0,0000010
0f32	rdmsr	2	0,0000010
0f33	rdpmc	2	0,0000010
0f41	cmovno	2	0,0000010
0f79	vmwrite	2	0,0000010
0fa1	pop	2	0,0000010
0fa6	xsha1	2	0,0000010
0fa6	xsha256	2	0,0000010
0fa7	xcrypt-cbc	2	0,0000010
0fa7	xcrypt-cfb	2	0,0000010

Код оп.	Мнемоника	Число	Доля, %
0fa7	xcrypt-ofb	2	0,0000010
0fa7	xstore-rng	2	0,0000010
0fae	fxrstor	2	0,0000010
0fae	fxsave	2	0,0000010
0fae	xsave	2	0,0000010
0fb9	ud1	2	0,0000010
0fc2	cmplepd	2	0,0000010
0fc7	xsavec	2	0,0000010
82/5	sub	2	0,0000010
c4	vcvtdq2pd	2	0,0000010
c4	vcvtdq2ps	2	0,0000010
c4	vcvtpd2ps	2	0,0000010
c4	vcvtps2dq	2	0,0000010
c4	vfmadd213sd	2	0,0000010
c4	vfmadd213ss	2	0,0000010
c4	vfmaddss	2	0,0000010
c4	vpminub	2	0,0000010
c4	vrcpps	2	0,0000010
c5	lds	2	0,0000010
c5	vcmplt_oqpd	2	0,0000010
c5	vcmpltps	2	0,0000010
c5	vcmpnleps	2	0,0000010
c5	vpsubsw	2	0,0000010
c5	vsqrtsd	2	0,0000010
d9	fldl2t	2	0,0000010
d9	fnop	2	0,0000010
d9	fprem1	2	0,0000010
db	frstpm(287	2	0,0000010

Код оп.	Мнемоника	Число	Доля, %
0f01	smsw	1	0,0000005
0f02	lar	1	0,0000005
0f03	lsl	1	0,0000005
0f09	wbinvd	1	0,0000005
0f0f	pfadd	1	0,0000005
0f0f	pfsub	1	0,0000005
0f11	movupd	1	0,0000005
0f7d	hsubps	1	0,0000005
0f91	setno	1	0,0000005
0fae	xsave64	1	0,0000005
0fb0	cmpxchg	1	0,0000005
0fb3	btr	1	0,0000005
0fb4	lfs	1	0,0000005
0fc2	cmpps	1	0,0000005
0fc3	movnti	1	0,0000005
82/6	xor	1	0,0000005
8f/1	vphaddwd	1	0,0000005
8f/1	vphaddwq	1	0,0000005
8f/1	vpmadcswd	1	0,0000005
8f/5	vpcomq	1	0,0000005
c4	kmovq	1	0,0000005
c4	vfnmaddps	1	0,0000005
c4	vmaxps	1	0,0000005
c4	vpacksswb	1	0,0000005
c4	vpaddusw	1	0,0000005
c4	vpmovsxdq	1	0,0000005
c5	vcvtsd2ss	1	0,0000005
c5	vcvtsi2ss	1	0,0000005

Код оп.	Мнемоника	Число	Доля, %
c5	vcvtss2sd	1	0,0000005
c5	vcvttss2si	1	0,0000005
c5	vmovlpd	1	0,0000005
c5	vsubss	1	0,0000005
d9	ftst	1	0,0000005
db	fndisi(8087	1	0,0000005

Общее чило инструкций: 207 778 299

ПРИЛОЖЕНИЕ 4. СОСТАВ И СИНОНИМЫ ИСПОЛЬЗУЕМЫХ ПРОЦЕССОРОМ ИНСТРУКЦИЙ

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
00 /r	ADD	reg/mem8, reg8	
01 /r	ADD	reg/mem64, reg64	
02 /r	ADD	reg8, reg/mem8	
03 /r	ADD	reg64, reg/mem64	
04 ib	ADD	AL, imm8	Да, на опкод 00
05 id	ADD	RAX, imm32	Да, на опкод 01
06	PUSH seg	–	
07	POP	ES	
08 /r	OR	reg/mem8, reg8	
09 /r	OR	reg/mem64, reg64	
0A /r	OR	reg8, reg/mem8	
0B /r	OR	reg64, reg/mem64	
0C ib	OR	AL, imm8	Да, на опкод 08
0D id	OR	RAX, imm32	Да, на опкод 09
0E	PUSH seg	–	
0F 00 /0	SLDT	mem16	
0F 00 /2	LLDT	–	
0F 00 /3	LTR	reg/mem16	
0F 01 /0	SGDT	mem16:64	
0F 01 /1	SIDT	mem16:64	
0F 01 /2	LGDT	mem16:64	
0F 01 /3	LIDT	mem16:64	
0F 01 /7	INVLPG	mem8	
0F 01 D9	VMMCALL	–	
0F 01 DA	VMLOAD	rAX	
0F 01 DB	VMSAVE	rAX	
0F 01 DC	STGI	–	
0F 01 DD	CLGI	–	
0F 01 DE	SKINIT	EAX	
0F 01 DF	INVLPGA	rAX, ECX	
0F 01 EE	RDPKRU	–	
0F 01 EF	WRPKRU	–	
0F 01 F8	SWAPGS	–	
0F 01 F9	RDTSCP	–	
0F 01 FD	RDPRU	–	
0F 01 FE	INVLPGB	–	
0F 03 /r	LSL	reg64, reg/mem16	
0F 05	SYSCALL	–	
0F 07	SYSRET	–	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
0F 08	INVD	–	
0F 09	WBINVD	–	
0F 0D /0	PREFETCH	mem8	
0F 0D /1	PREFETCHW	mem8	
0F 18 /0	PREFETCHNTA	mem8	
0F 18 /1	PREFETCHT0	mem8	
0F 18 /2	PREFETCHT1	mem8	
0F 18 /3	PREFETCHT2	mem8	
0F 21 /r	MOV	reg64, DRn	
0F 23 /r	MOV	DRn, reg64	
0F 30	WRMSR	–	
0F 31	RDTSC	–	
0F 32	RDMSR	–	
0F 33	RDPMC	–	
0F 34	SYSENTER	–	
0F 35	SYSEXIT	–	
0F 38 F0 /r	MOVBE	reg64, mem64	
0F 38 F1 /r	MOVBE	mem64, reg64	
0F 40 /r	CMOVO	reg32, reg/mem32	
0F 41 /r	CMOVNO	reg32, reg/mem32	
0F 42 /r	CMOVC	reg32, reg/mem32	
0F 43 /r	CMOVAE	reg32, reg/mem32	
0F 43 /r	CMOVNB	reg32,reg/mem32	
0F 44 /r	CMOVE	reg32, reg/mem32	
0F 45 /r	CMOVNE	reg32, reg/mem32	
0F 46 /r	CMOVNA	reg32, reg/mem32	
0F 47 /r	CMOVA	reg32, reg/mem32	
0F 48 /r	CMOVS	reg32, reg/mem32	
0F 49 /r	CMOVNS	reg32, reg/mem32	
0F 4A /r	CMOVPE	reg32, reg/mem32	
0F 4B /r	CMOVPO	reg32, reg/mem32	
0F 4C /r	CMOVNGE	reg32, reg/mem32	
0F 4D /r	CMOVGE	reg32, reg/mem32	
0F 4E /r	CMOVNG	reg32, reg/mem32	
0F 4F /r	CMOVG	reg32, reg/mem32	
0F 6E /r	MOVD	mmx, reg/mem64	
0F 7E /r	MOVD	reg/mem64, mmx	
0F 80 cw	JO	rel16off	
0F 81 cw	JNO	rel16off	
0F 82 cw	JNAE	rel16off	
0F 83 cw	JAE	rel16off	
0F 84 cw	JE	rel16off	
0F 85 cw	JNE	rel16off	
0F 86 cw	JNA	rel16off	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
0F 87 cw	JA	rel16off	
0F 88 cw	JS	rel16off	
0F 89 cw	JNS	rel16off	
0F 8A cw	JPE	rel16off	
0F 8B cw	JPO	rel16off	
0F 8C cw	JNGE	rel16off	
0F 8D cw	JGE	rel16off	
0F 8E cw	JNG	rel16off	
0F 8F cw	JG	rel16off	
0F 90 /0	SETO	reg/mem8	
0F 91 /0	SETNO	reg/mem8	
0F 92 /0	SETC	reg/mem8	
0F 93 /0	SETNC	reg/mem8	
0F 94 /0	SETZ	reg/mem8	
0F 95 /0	SETNZ	reg/mem8	
0F 96 /0	SETBE	reg/mem8	
0F 97 /0	SETNBE	reg/mem8	
0F 98 /0	SETS	reg/mem8	
0F 99 /0	SETNS	reg/mem8	
0F 9A /0	SETPE	reg/mem8	
0F 9B /0	SETPO	reg/mem8	
0F 9C /0	SETNGE	reg/mem8	
0F 9D /0	SETNG	reg/mem8	
0F 9E /0	SETGE	reg/mem8	
0F 9F /0	SETNLE	reg/mem8	
0F A0	PUSH seg	–	
0F A1	POP	FS	
0F A2	CPUID	–	
0F A3 /r	BT	reg/mem64, reg64	
0F A4 /r ib	SHLD	reg/mem64, reg64, imm8	Да, зануление двух старших бит непосредственного операнда
0F A5 /r	SHLD	reg/mem64, reg64, CL	
0F A8	PUSH seg	–	
0F A9	POP	GS	
0F AB /r	BTS	reg/mem64, reg64	
0F AC /r ib	SHRD	reg/mem64, reg64, imm8	Да, зануление двух старших бит непосредственного операнда
0F AD /r	SHRD	reg/mem64, reg64, CL	
0F AE E8	LFENCE	–	
0F AE F0	MFENCE	–	
0F AE F8	SFENCE	–	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
0F AF /r	IMUL	reg64, reg/mem64	
0F B0 /r	CMPXCHG	reg/mem8, reg8	
0F B1 /r	CMPXCHG	reg/mem64, reg64	
0F B2 /r	LSS	reg32, mem16:32	
0F B3 /r	BTR	reg/mem64, reg64	
0F B4 /r	LFS	reg32, mem16:32	
0F B5 /r	LGS	reg32, mem16:32	
0F B6 /r	MOVZX	reg64, reg/mem8	
0F B7 /r	MOVZX	reg64, reg/mem16	
0F BA /4 ib	BT	reg/mem64, imm8	Да, замена на 0F A3
0F BA /5 ib	BTS	reg/mem64, imm8	Да, замена на 0F AB
0F BA /6 ib	BTR	reg/mem64, imm8	Да, замена на 0F B3
0F BA /7 ib	BTC	reg/mem64, imm8	Да, замена на 0F BB
0F BB /r	BTC	reg/mem64, reg64	
0F BC /r	BSF	reg64, reg/mem64	
0F BD /r	BSR	reg64, reg/mem64	
0F C0 /r	XADD	reg/mem8, reg8	
0F C1 /r	XADD	reg/mem64, reg64	
0F C3 /r	MOVNTI	mem64, reg64	
0F C7 /1 m128	CMPXCHG16B	mem128	
0F C7 /6	RDRAND	reg64	
0F C8 +rq	BSWAP	reg64	
10 /r	ADC	reg/mem8, reg8	
11 /r	ADC	reg/mem64, reg64	
12 /r	ADC	reg8, reg/mem8	
13 /r	ADC	reg64, reg/mem64	
14 ib	ADC	AL, imm8	Да, на опкод 10
15 id	ADC	RAX, imm32	Да, на опкод 11
16	PUSH seg	–	
17	POP	SS	
18 /r	SBB	reg/mem8, reg8	
19 /r	SBB	reg/mem64, reg64	
1A /r	SBB	reg8, reg/mem8	
1B /r	SBB	reg64, reg/mem64	
1C ib	ADC	AL, imm8	Да, на опкод 18
1D id	ADC	RAX, imm32	Да, на опкод 19
1E	PUSH seg	–	
1F	POP	DS	
20 /r	AND	reg/mem8, reg8	
21 /r	AND	reg/mem64, reg64	
22 /r	AND	reg8, reg/mem8	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
23 /r	AND	reg64, reg/mem64	
24 ib	AND	AL, imm8	Да, на опкод 20
25 id	AND	RAX, imm32	Да, на опкод 21
28 /r	SUB	reg/mem8, reg8	
29 /r	SUB	reg/mem64, reg64	
2A /r	SUB	reg8, reg/mem8	
2B /r	SUB	reg64, reg/mem64	
2C ib	SUB	AL, imm8	Да, на опкод 28
2D id	SUB	RAX, imm32	Да, на опкод 29
2F	DAS	–	
30 /r	XOR	reg/mem8, reg8	
31 /r	XOR	reg/mem64, reg64	
32 /r	XOR	reg8, reg/mem8	
33 /r	XOR	reg64, reg/mem64	
34 ib	XOR	AL, imm8	Да, на опкод 30
35 id	XOR	RAX, imm32	Да, на опкод 31
37	AAA	–	
38 /r	TEST	reg/mem8, reg8	
39 /r	TEST	reg/mem64, reg64	
3A /r	TEST	reg8, reg/mem8	
3B /r	TEST	reg64, reg/mem64	Да, на опкод 38
3C ib	TEST	AL, imm8	Да, на опкод 39
3F	AAS	–	
50 +rq	PUSH	reg64	
58 +rq	POP	reg64	
60	PUSHAD	–	
61	POPAD	–	
62 /r	BOUND	reg32, mem32&mem32	
63 /r	MOVSXD	reg64, reg/mem32	
66 0F 38 F6 /r	ADCX	reg64, reg/mem64	
66 0F 50 /r	MOVMSKPD	reg32, xmm	
66 0F 6E /r	MOVD	xmm, reg/mem64	
66 0F 7E /r	MOVD	reg/mem64, xmm	
66 0F AE /6	CLWB	–	
66 0F AE /7	CLFLUSHOPT	mem8	
68 id	PUSH	imm64	Да, на опкод с 50 по 57
69 /r ib	IMUL	reg64, reg/mem64, imm32	Да, на опкод 0F AF
6A ib	PUSH	imm8	Да, на опкод с 50 по 57
6B /r ib	IMUL	reg64, reg/mem64, imm8	Да, на опкод 0F AF
6C	INSB	–	
6D	INS	mem32, DX	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
6E	OUTS	DX, mem8	
6F	OUTS	DX, mem16	
6F	OUTS	DX, mem32	
70 cb	JO	rel8off	
71 cb	JNO	rel8off	
72 cb	JNAE	rel8off	
73 cb	JAE	rel8off	
74 cb	JE	rel8off	
75 cb	JNE	rel8off	
76 cb	JNA	rel8off	
77 cb	JA	rel8off	
78 cb	JS	rel8off	
79 cb	JNS	rel8off	
7A cb	JPE	rel8off	
7B cb	JPO	rel8off	
7C cb	JNGE	rel8off	
7D cb	JGE	rel8off	
7E cb	JNG	rel8off	
7F cb	JG	rel8off	
80 /0 ib	ADD	reg/mem8, imm8	Да, на опкод 00
80 /1 ib	OR	reg/mem8, imm8	Да, на опкод 08
80 /2 ib	ADC	reg/mem8, imm8	Да, на опкод 10
80 /3 ib	SBB	reg/mem8, imm8	Да, на опкод 18
80 /4 ib	AND	reg/mem8, imm8	Да, на опкод 20
80 /5 ib	SUB	reg/mem8, imm8	Да, на опкод 28
80 /6 ib	XOR	reg/mem8, imm8	Да, на опкод 30
80 /7 ib	CMP	reg/mem8, imm8	Да, на опкод 38
81 /0 id	ADD	reg/mem64, imm32	Да, на опкод 01
81 /1 id	OR	reg/mem64, imm32	Да, на опкод 09
81 /2 id	ADC	reg/mem64, imm32	Да, на опкод 11
81 /3 id	SBB	reg/mem64, imm32	Да, на опкод 19
81 /4 id	AND	reg/mem64, imm32	Да, на опкод 21
81 /5 id	SUB	reg/mem64, imm32	Да, на опкод 29
81 /6 id	XOR	reg/mem64, imm32	Да, на опкод 31
81 /7 id	CMP	reg/mem64, imm32	Да, на опкод 39
83 /0 ib	ADD	reg/mem64, imm8	Да, на опкод 01
83 /1 ib	OR	reg/mem64, imm8	Да, на опкод 09
83 /2 ib	ADC	reg/mem64, imm8	Да, на опкод 11
83 /3 ib	SBB	reg/mem64, imm8	Да, на опкод 19
83 /4 ib	AND	reg/mem64, imm8	Да, на опкод 21
83 /5 ib	SUB	reg/mem64, imm8	Да, на опкод 29
83 /6 ib	XOR	reg/mem64, imm8	Да, на опкод 31
83 /7 ib	CMP	reg/mem64, imm8	Да, на опкод 39
84 /r	TEST	reg/mem8, reg8	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
85 /r	TEST	reg/mem64, reg64	
86 /r	XCHG	reg8, reg/mem8	
87 /r	XCHG	reg64, reg/mem64	
88 /r	MOV	reg/mem8, reg8	
89 /r	MOV	reg/mem64, reg64	
8A /r	MOV	reg8, reg/mem8	
8B /r	MOV	reg64, reg/mem64	
8E /r	MOV	segReg, reg/mem16	
8F /0	POP	reg/mem64	
8F/0A 10 /r id	BEXTR	reg64, reg/mem64, imm32	Да, на опкод C4/02 F7
8F/0A 12 /0 id	LWPINS	imm32	Да, не требуется – не используется
8F/0A 12 /1 id	LWPVAL	imm32	
90 +rq	XCHG	reg64, RAX	
98	CBW	–	
99	CWD	–	
9A cp	CALL	FAR pntr16:32	
9D	POPFQ	–	
9E	SAHF	–	
9F	LAHF	–	
A0	MOV	AL, moffset8	
A1	MOV	RAX, moffset64	
A2	MOV	moffset8, AL	
A3	MOV	moffset64, RAX	
A4	MOVSB	mem8, mem8	
A5	MOVS	mem64, mem64	
A6	CMPS	mem8, mem8	
A7	CMPS	mem16, mem16	
A8 ib	TEST	AL, imm8	Да, на опкод 84
A9 id	TEST	RAX, imm32	Да, на опкод 85
AA	STOS	mem8	
AB	STOS	mem16	
AC	LODS	mem8	
AD	LODS	mem16	
AE	SCAS	mem8	
AF	SCAS	mem64	
B0 +rb ib	MOV	reg8, imm8	Да, представление операнда в защищенном виде
B8 +rq iq	MOV	reg64, imm64	
C0 /1 ib	ROR	reg/mem8, imm8	Да, зануление двух старших бит непосредственного операнда
C0 /2 ib	RCL	reg/mem8, imm8	
C0 /3 ib	RCR	reg/mem8,imm8	
C0 /4 ib	SHL	reg/mem8, imm8	
C0 /5 ib	SHR	reg/mem8, imm8	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
C0 /6 ib	SHL	reg/mem8, imm8	
C0 /7 ib	SAR	reg/mem8, imm8	
C1 /1 ib	ROR	reg/mem64, imm8	
C1 /2 ib	RCL	reg/mem64, imm8	
C1 /3 ib	RCR	reg/mem64, imm8	
C1 /4 ib	SHL	reg/mem64, imm8	
C1 /5 ib	SHR	reg/mem64, imm8	
C1 /6 ib	SHL	reg/mem64, imm8	
C1 /7 ib	SAR	reg/mem64, imm8	
C2 iw	RET	imm16	Да, вставка защитного кода перед
C3	RET	–	
C4/03 F0 /r	RORX	reg64, reg/mem64, imm8	Да, зануление двух старших бит непосредственного операнда
C6 /0 ib	MOV	reg/mem8, imm8	Да, на опкод 88
C7 /0 id	MOV	reg/mem64, imm32	Да, на опкод 89
C8 iw ib	ENTER	imm16, imm8	Да, не требуется – не используется
C9	LEAVE	–	
CA iw	RETF	imm16	Да, не требуется – не используется
CB	RETF	–	
CD ib	INT	imm8	Да, привилегир.
CE	INTO	–	
CF	IRET	–	
D0 /1	ROR	reg/mem8, 1	
D0 /2	RCL	reg/mem8,1	
D0 /3	RCR	reg/mem8, 1	
D0 /4	SAL	reg/mem8, 1	
D0 /4	SHL	reg/mem8, 1	
D0 /5	SHR	reg/mem8, 1	
D0 /7	SAR	reg/mem8, 1	
D1 /1	ROR	reg/mem64, 1	
D1 /2	RCL	reg/mem64, 1	
D1 /3	RCR	reg/mem64,1	
D1 /4	SHL	reg/mem64, 1	
D1 /5	SHR	reg/mem64, 1	
D1 /6	SHR	reg/mem64, 1	
D1 /7	SAR	reg/mem64, 1	
D2 /1	ROR	reg/mem8, CL	
D2 /2	RCL	reg/mem8, CL	
D2 /3	RCR	reg/mem8,CL	
D2 /4	SHL	reg/mem8, CL	
D2 /5	SHR	reg/mem8, CL	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
D2 /6	SHR	reg/mem8, CL	
D2 /7	SAR	reg/mem8, CL	
D3 /1	ROR	reg/mem64, CL	
D3 /2	RCL	reg/mem64, CL	
D3 /3	RCR	reg/mem64,CL	
D3 /4	SHL	reg/mem32, CL	
D3 /5	SHR	reg/mem64, CL	
D3 /5	SHД	reg/mem64, CL	
D3 /7	SAR	reg/mem64, CL	
D4 0A	AAM	–	
D5 0A	AAD	–	
E0 cb	LOOPNE	rel8off	
E1 cb	LOOPE	rel8off	
E2 cb	LOOP	rel8off	
E3 cb	JRCXZ	rel8off	
E4 ib	IN	AL, imm8	Да, привилегир.
E5 ib	IN	EAX, imm8	
E6 ib	OUT	imm8, AL	
E7 ib	OUT	imm8, EAX	
E8 id	CALL	rel32off	
E9 cd	JMP	rel32off	
EA cp	JMP	FAR pntr16:32	
EB cb	JMP	rel8off	
EC	IN	AL, DX	
ED	IN	EAX, DX	
EE	OUT	DX, AL	
EF	OUT	DX, EAX	
F2 0F 01 FE	RMPUPDATE	–	
F2 0F 01 FF	PVALIDATE	–	
F2 0F 38 F0 /r	CRC32	reg32, reg/mem8	
F2 0F 38 F1 /r	CRC32	reg32, reg/mem32	
F2 REX 0F 38 F0 /r	CRC32	reg32, reg/mem8	
F2 REX.W 0F 38 F0 /r	CRC32	reg64, reg/mem8	
F2 REX.W 0F 38 F1 /r	CRC32	reg64, reg/mem64	
F3 0F 01 FA	MCOMMIT	–	
F3 0F 01 FE	RMPADJUST	–	
F3 0F 09	WBNOINVD	–	
F3 0F 38 F6 /r	ADOX	reg64, reg/mem64	
F3 0F AE /2	WRFSBASE	reg64	
F3 0F AE /3	WRGSBASE	reg64	
F3 0F BC /r	TZCNT	reg64, reg/mem64	
F3 0F BD /r	LZCNT	reg32, reg/mem32	
F3 0F C7/7	RDPID	–	
F4	HLT	–	

Формат	Мнемоника	Аргументы	Возможность опасного значения и замена
F6 /0 ib	TEST	reg/mem8, imm8	Да, на опкод 84
F6 /1 iq	TEST	reg/mem64, imm32	
F6 /2	NOT	reg/mem8	
F6 /4	MUL	reg/mem8	
F6 /5	IMUL	reg, mem8	
F6 /6	DIV	reg/mem8	
F6 /7	IDIV	reg/mem8	
F7 /0 id	TEST	reg/mem8, imm6	Да, на опкод 85
F7 /1 id	TEST	reg/mem64, imm32	
F7 /2	NOT	reg/mem64	
F7 /4	MUL	reg/mem64	
F7 /5	IMUL	reg/mem64	
F7 /6	DIV	reg/mem16	
F7 /7	IDIV	reg/mem64	
F8	CLC	–	
FA	CLI	–	
FB	STI	–	
FC	CLD	–	
FD	STD	–	
FE /0	INC	reg/mem8	
FF /1	DEC	reg/mem64	
FF /2	CALL	reg/mem64	
FF /3	CALL	FAR mem16:32	
FF /4	JMP	reg/mem64	
FF /5	JMP	FAR mem16:32	
FF /6	PUSH	reg/mem64	

ПРИЛОЖЕНИЕ 5. РЕЗУЛЬТАТЫ ОБРАБОТКИ ЗАЩИЩАЕМЫХ ПРИЛОЖЕНИЙ

Приложение	Уникальные гаджеты			Регистры, определяемые атакующим			Оперируемые регистры			Дост. арг., $N\pi$ (полностью/частично)		
	Ориг.	G-free	zpk	Ориг.	G-free	zpk	Ориг.	G-free	zpk	Ориг.	G-free	zpk
Минималистическое приложение, ур. оптимиз. 0	45	15	3	eax, r12, r13, r14, r15, rbp, rdi, rsi, rsp	dil	Нет	ebx, cl	al, bh	al	2/3	0/1	0
Минималистическое приложение, ур. оптимиз. 2	38	12	3	al, r12, r13, r14, r15, rbp, rdi, rsi, rsp	di1	Нет	eax, ebx, cl	al	al	2/3	0/1	0
Синтетический тест, нет оптимиз.	389		3	rax, ch, rbx, edx, rsi, rdi, rbp, rsp, r8d, r12, r13, r14, r15		Нет	rcx, rdx, fr7		al	4/5		0
Синтетический тест, ур. оптимиз. 2	122		3	eax, rbx, rsi, rdi, rbp, rsp, r12, r13, r14, r15		Нет	rax, ecx, edx		al	2/4		0
Учебное уязвимое приложение [104]	85		3	eax, rbx, rsi, rdi, rbp, rsp		Нет	rax, edx		al	2/2		0
hostname	88		3	rax, r12, r13, r14, r15, rbp, rbx, rdi, rsi, rsp		Нет	r8d		al	2/2		0
mountpoint	76		3	r12, r13, r14, r15, rbp, rbx, rdi, rsi, rsp		Нет	eax		al	2/2		0
coremark	232	34	4	rax, rbx, rsi, rdi, rbp, rsp, r8b, r12, r13, r14, r15, xmm0	edi	bh	ecx, edx, r9d	eax, cl, xmm9	eax	2/6	0/1	0
ls	1159		7	rax, r12, r13, r14, r15, rbp, rbx, rdi, rsi, rsp, rcx, rdx		Нет	fr0, fr3		al	4/4		0
asn1c	1467	272	10	rsi, rdi, rdx, rcx, r8, rax, rbp, rsp, rbx, r12, r13 ,r14, r15	rsi, rdi, rdx, rcx, rbx, rbp, rsp, rax, r12	eax	r10, fr0, fr5, xmm3	r9b, fr0	rax	5/5	4/4	0
sshd	4713		23	eax, rbx, rsi, rdi, rbp, rsp, r8, r11, r12, r13, r14, r15		eax, edx	r9, r10		rax	6/6		0

ПРИЛОЖЕНИЕ 6. ПРОГРАММНЫЙ КОД ДЕМОНСТРИРУЮЩЕГО УЯЗВИМОСТИ ПРИЛОЖЕНИЯ И ЭКСПЛОЙТА ДЛЯ НЕГО

Код уязвимого демострационного приложения:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
enum Commands_e {cmdExit= 0, cmdRead= 1, cmdWrite= 2};
int nested_proc(int tcp_sock) {
  for (;;) {
    int32_t value[4]= {0,0,0,0};
    int read_result= read(tcp_sock, value, 1024);
    if (0 >= read_result) return read_result;
    if (cmdExit == value[0]) break; // Ok
    if (cmdRead == value[0]) write(tcp_sock, &value[2], value[1]);
    if (cmdWrite == value[0]) <запись во внутренние структуры>
  }
  return 0;
}

int main(int , char **) {
  sockaddr_in bind_point= {AF_INET,0x5555,{INADDR_ANY},0};
  int server_sock= socket(AF_INET, SOCK_STREAM, 0);
  bind(server_sock, reinterpret_cast<const sockaddr *>\
   (&bind_point), sizeof(bind_point));
  listen(server_sock, 1);
  for (;;) {
    int client_socket= accept(server_sock, NULL, NULL);
    if (nested_proc(client_socket) < 0) break;
    close(client_socket);
  }
  return 0;
}
```

ПРИЛОЖЕНИЕ 7. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021666279

Библиотека защиты программного кода (libzpk)

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева» (СибГУ им. М.Ф. Решетнева) (RU)***

Автор(ы): ***Лубкин Иван Александрович (RU)***

Заявка № **2021665560**
Дата поступления **04 октября 2021 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***12 октября 2021 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012615882

Программное средство защиты бинарного кода от исследования

Правообладатель(ли): ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева» (СибГАУ) (RU)***

Автор(ы): ***Шудрак Максим Олегович, Лубкин Иван Александрович, Золотарев Вячеслав Владимирович (RU)***

Заявка № **2012613822**

Дата поступления **11 мая 2012 г.**

Зарегистрировано в Реестре программ для ЭВМ

27 июня 2012 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

ПРИЛОЖЕНИЕ 8. АКТЫ ВНЕДРЕНИЯ

РТК-СИБИРЬ

660062, Красноярск, ул. Телевизорная 1, строение 9, а/я 8650,
тел.: +7(391) 2-903-160, факс: +7(391) 2-903-161, e-mail: office@rtksib.ru

Исх. № _____
Дата «____» _____ 20____г.

На № _____ от _____ 20____г.

АКТ ВНЕДРЕНИЯ

результатов диссертационной работы Лубкина И.А. в производственный процесс

Настоящим актом подтверждается, что результаты диссертационной работы Лубкина И. А. на тему «Метод снижения подверженности приложений к реализации уязвимостей за счет обфускации машинного кода» внедрены в процесс эксплуатации программного обеспечения АО «РТК-Сибирь» в части защиты от уязвимостей сервисов, доступных через Интернет. Предложенные решения применены в рамках:

1. Отработки метода на основе тестового сетевого приложения, в которое были намеренно внесены заведомо известные уязвимости (примитив чтения, позволяющий обойти ASLR и уязвимость переполнения буфера). Далее был написан эксплойт, в котором использовалась *RoP*-цепочка гаджетов. После этого было выполнено защитное преобразование уязвимого приложения согласно предложенному методу. Использование предложенного метода оценки эффективности систем защиты программ от *RoP*-атак показало, что атака не может быть реализована. Данный результат подтвержден тем, что средство ROPgadget перестало обнаруживать пригодные для атаки участки, что подтверждает корректность предложенного метода оценки.
2. Защитного преобразования эксплуатируемого сервиса sshd из состава AstraLinux 1.6 SE, для которого отсутствуют исходные коды. Для обработанного приложения была проверена работоспособность и отсутствие участков, пригодных для атаки.

В результате внедрения программного средства, разработанного Лубкиным И. А., было произведено применение защиты от уязвимостей для программного средства, к которым имеет доступ атакующий через сеть Интернет.

Генеральный директор В.И. Тихонов

УТВЕРЖДАЮ
Проректор по научной работе
ФГАОУ ВО «Сибирский федеральный университет»
_____ Барышев Р. А.
«___» _____ 2022 г.

АКТ

об использовании результатов диссертационной работы Лубкина И. А. «Метод снижения подверженности приложений к реализации уязвимостей за счет обфускации машинного кода»

Настоящим актом подтверждается использование результатов диссертационной работы Лубкина Ивана Александровича в ФГАОУ ВО «Сибирский федеральный университет».

Лубкиным И. А. разработан модифицированный метод снижения числа пригодных для проведения RoP-атак участков в программах и модифицированная методика контроля целостности графа потока управления.

Результаты, полученные в ходе выполнения диссертационной работы, использовались в научно-исследовательской части в рамках работ по устранению уязвимостей в разрабатываемом программном обеспечении для опытно-конструкторской работы «Разработка информационно-вычислительного комплекса измерительной станции БИС-Н» в рамках договора № 12201871148420101280016 98/722-20/18 от 01.02.2018.

Руководитель департамента науки и инновационной деятельности СФУ _____ Казаков В. С.

Руководитель темы _____ Саломатов Ю. П.

МИНИСТЕРСТВО НАУКИ И
ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное
образовательное учреждение
высшего образования
**«Сибирский государственный
университет науки и технологий
имени академика М.Ф. Решетнева»
(СибГУ им. М.Ф. Решетнева)
Институт информатики и
телекоммуникаций
(ИИТК)**
просп. им. газеты «Красноярский рабочий», 31
г. Красноярск, 660014
Тел.: (391) 264-00-14. Факс: (391) 264-47-09
E-mail: info@sibsau.ru
http://www.sibsau.ru
ОКПО 02069734, ОГРН 1022402056038
ИНН/КПП 2462003320/246201001
30.08.2022 № 142/6

Акт внедрения

Результаты, полученные в ходе проведения Лубкиным И. А. диссертационного исследования в рамках гранта Минобрнауки России № 21/2020 на 2020–2021 гг. «Метод снижения подверженности приложений к реализации уязвимостей за счет обфускации машинного кода» (автор является единоличным исполнителем) были внедрены в учебный процесс в следующем объеме:

1. Внедрение результатов диссертационной работы в учебный процесс института информатики и телекоммуникаций для направлений подготовки 10.03.01 «Информационная безопасность», 10.05.03 «Информационная безопасность автоматизированных систем», 10.05.02 «Информационная безопасность телекоммуникационных систем» в рамках предметов «Аппаратные средства вычислительной техники», «Организация ЭВМ и вычислительных систем» и «Безопасность операционных систем».

2. Комплект практических заданий для указанных выше предметов, демонстрирующий методы проведения *RoP*-атаки и методы защиты от них при различных возможностях атакующего. Предложенный в диссертационной работе метод демонстрирует защищенность от рассматриваемых уязвимостей.

Подготовлены задания для соревнований по практическим аспектам информационной безопасности в формате «Capture the Flag» для школьников и студентов, запланированных в 2022 году.

Директор ИИТК А.М. Попов

ПРИЛОЖЕНИЕ 9. ДИПЛОМЫ И НАГРАДЫ

ДИПЛОМ

I степени

Второй всероссийской молодежной конференции
по проблемам информационной безопасности

ПЕРСПЕКТИВА-2012

присуждается

Лубкину Ивану Александровичу

Шудраку Максиму Олеговичу

за доклад «Методика декомпиляции бинарного кода и ее применение в сфере информационной безопасности»

Председатель оргкомитета
конференции, д.т.н.,
профессор Макаревич О.Б.

Таганрог
2012

Диплом

III степени
присуждается

Лубкину Ивану Александровичу
студенту
занявшему 3 место в Седьмом Всероссийском конкурсе студентов и аспирантов по информационной безопасности "SIBINFO-2007"за доклад на тему "Защита программного кода посредством хеширования адресов перехода"

Зам. председателя , доктор технических наук,
Заведующий кафедрой Комплексной
информационной безопасности электронно-
вычислительных систем
Томского государственного университета
систем управления и радиоэлектроники

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственный университет систем управления и радиоэлектроники
Структурное подразделение
ЦЕНТР ТЕХНОЛОГИЙ БЕЗОПАСНОСТИ
КАФЕДРА КИБЭВС
г. Томск

А.А. Шелупанов

18.апреля 2007г.
г.Томск