Федеральное государственное бюджетное учреждение науки
Санкт-Петербургский институт информатики и автоматизации
Российской академии наук
(СПИИРАН)

На правах рукописи

Салахутдинова Ксения Иркиновна

Методика идентификации исполняемых файлов на основе статического анализа характеристик дизассемблированного кода программ

Специальность 05.13.19 – Методы и системы защиты информации, информационная безопасность

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
доктор технических наук, профессор
Лебедев Илья Сергеевич

Санкт-Петербург – 2019

Оглавление

Введение 6

Глава 1. Постановка задачи обеспечения безопасности автоматизированной системы от потенциальных угроз со стороны несанкционированно установленного программного обеспечения 15

1.1 Обзор проблематики современных подходов к идентификации программного обеспечения 15

1.2 Модель угроз информационной безопасности при обработке информации конфиденциального характера в информационной системе 18

1.3 Модель вероятного нарушителя информационной безопасности и оценка его возможностей 22

Выводы по главе 1 25

Глава 2. Обзор существующих методов идентификации программного обеспечения 27

2.1 Методы идентификации, основанные на статическом характере анализа характеристик программного обеспечения 29

2.2 Методы идентификации, основанные на динамическом характере сбора характеристик программного обеспечения 37

2.3 Методы идентификации, основанные на сборе информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента 39

2.4 Постановка задачи исследования 48

Выводы по главе 2 52

Глава 3. Разработка методики идентификации установленного программного обеспечения на основе статических характеристик программного кода файла 54

3.1 Анализ структуры и характеристик исполняемого файла 55

3.1.1 Представление исходного кода на языке ассемблера 59

3.1.2 Анализ особенностей ассемблерного кода программного обеспечения 61

3.2 Подход к представлению программного обеспечения..... 63

3.2.1 Модель представления исполняемого файла..... 65

3.2.2 Идентификационное признаковое пространство 66

3.2.3 Модель представления программного обеспечения 68

3.3 Метод формирования сигнатур исполняемых файлов, обладающий достаточной гибкостью по распознаванию различных версий программ, а также позволяющий наиболее точно производить идентификацию..... 69

3.3.1 Критерии и алгоритм отбора наиболее информативных признаков 70

3.3.2 Формирование унифицированных сигнатур программ (УСП), используемого в дальнейшем в статистическом методе сравнения сигнатур и с использованием искусственной нейронной сети..... 74

3.3.3 Формирование сигнатур программ (СП), используемого в дальнейшем в методе сравнения сигнатур на основе градиентного бустинга деревьев решений..... 81

3.4 Метод сравнения сигнатуры идентифицируемого исполняемого файла (СИИФ) с эталонной сигнатурой программы (ЭСП) 82

3.4.1 Статистические методы сравнения сигнатур на основе критерия однородности хи-квадрат и критерия Колмогорова..... 83

3.4.2 Метод сравнения сигнатур на основе машинного обучения..... 86

3.4.3 Оценка вычислительной сложности различных методов сравнения СИИФ с ЭСП 91

3.5 Методика идентификации программного обеспечения на основе статических характеристик программного кода файла 92

3.5.1 Предварительный этап методики идентификации программного обеспечения. Сбор и формирование обучающей выборки и составление АЭСП, содержащий ЭСП или унифицированные ЭСП 94

3.5.2 Основные этапы методики идентификации исполняемого файла ... 96

3.6 Оценка точности идентификации программного обеспечения 98

3.7 Ограничения методики 100

Выводы по главе 3 100

Глава 4. Экспериментальная проверка точности идентификации программного обеспечения и анализ полученных результатов 102

4.1 Входные данные экспериментов. Обучающая и тестовая выборки 102

4.2 Определение точности идентификации при использовании разработанных методов сравнения СИИФ с ЭСП 103

4.2.1 Точность идентификации при использовании статистических методов сравнения сигнатур 103

4.2.2 Точность идентификации при использовании метода сравнения сигнатур на основе машинного обучения 111

4.2.3 Повышение точности идентификации путем использования аддитивного критерия Фишберна 117

4.3 Определение итоговой точности идентификации программного обеспечения на основе предложенной методики 120

4.4 Использование результатов исследования для повышения информационной безопасности автоматизированных систем 123

Выводы по главе 4 126

Заключение 128

Список сокращений и условных обозначений 131

Список использованной литературы 133

Приложение 1. Значения информативности для 118 ассемблерных команд 142

Приложение 2. Числовые идентификаторы эталонных программ 143

Приложение 3. Коды сравнения сигнатур с помощью библиотек градиентного бустинга деревьев решений..... 145

Приложение 4. Свидетельство о регистрации программы для ЭВМ..... 150

Приложение 5. Копии актов внедрения..... 156

Приложение 6. Публикации соискателя по теме Диссертации 159

Введение

Актуальность работы. Свободно распространяемое программное обеспечение (ПО) является неотъемлемой частью современных информационных систем, эксплуатируемых в различных секторах экономики, оно позволяет расширить возможности по осуществлению процессов анализа, управления, принятия решений.

Применение открытого ПО обуславливает необходимость разработки дополнительных методов, систем и средств защиты информации. Возможные дефекты программного обеспечения, наличие не декларированных возможностей, нелегальное использование интеллектуальной собственности, применение специальных программ, направленных на преодоление установленной защиты, могут привести к росту числа уязвимостей и повлиять на информационную безопасность систем.

В связи с этим возникает необходимость решения ряда задач идентификации, верификации и валидации программного обеспечения. Предлагаемые решения ориентированы, в основном, на отслеживание фиксированного состояния кода программ на носителях и в оперативной памяти, что не всегда позволяет оперативно определить санкционированные модификации, изменения версий.

Данная работа направлена на решение задачи идентификации программного обеспечения, в условии изменения версий распространяемого ПО, на основе процедуры построения информативной модели в виде математического кортежа по выбранному признаковому пространству, характеристики которой позволяют найти однозначное соответствие между анализируемой последовательностью и хранящимся эталоном исполняемого файла.

Степень разработанности темы. Различным подходам идентификации программного обеспечения посвятили свои работы такие отечественные ученые как Казарин О. В., Копыльцов А.В., Сорокин И.В., Антонов А. Е., Федулов А. С., зарубежные – Kornblum J.D., Long C., Ebringer T., Sun L., Boztas S., Schultz M. G., Eskin E., Santos I. и другие.

Существующие подходы к идентификации исполняемых файлов, по способу анализа характеристик программы, можно разделить на статические и динамические. К динамическим относятся методы, рассматривающие программу как процесс выполнения некоторого алгоритма или как последовательную смену состояний на графе переходов программы. Статические, в свою очередь, делятся на не сигнатурные (методы проверки целостности): сравнение контрольных сумм, с полной копией данных, вычисление CRC (Cyclic Redundancy Check), хэш-сумм, использование имитовставки, цифровой подписи; и сигнатурные (методы сравнения признаковых последовательностей): выделение «магического числа», сопоставление характерных последовательностей байтов и др.

В качестве информативных признаков файла в отечественных и зарубежных научных работах рассматриваются такие признаки как операторы в тексте программ (Казарин О. В.), энтропия сегментов файлов, цифровые изображения (Копыльцов А.В., Сорокин И.В.), блоки в сегментированной программе (Антонов А.Е., Федулов А.С.), хеш-функции и кусочно-зависимое хеширование (Kornblum J.D., Long C., Guoyin W., Baier H., Frank B.), статистические профили (Ebringer T., Sun L., Boztas S.), бинарные профили, последовательность строк, шестнадцатеричные дампы (Schultz M. G., Eskin E., Zadok F., Stolfo S. J.), *n*-граммы (Santos I., Brezo F., Ugarte-Pedrero X., Bringas P. G.) и многие другие.

Таким образом, существующие подходы к идентификации установленного программного обеспечения обладают рядом ограничений и недостаточными возможностями по обеспечению комплексной реализации мер по информационной безопасности.

Рассматриваемая в работе методика идентификации программного обеспечения направлена на распознавание программы, не основываясь на ее целостности. В результате создается достаточная гибкость, позволяющая определять исполняемые файлы, ранее не задействованные в процессе формирования эталонных сигнатур. Процесс идентификации сравнивает две сигнатуры: эталонную – созданную на основе обучающей выборки и сигнатуру

идентифицируемого исполняемого файла – создаваемую непосредственно перед этапом сравнения.

Всевозрастающее количество версий программного обеспечения, обуславливает сложность полноценной комплексной защиты информации. В следствии чего, возникает необходимость в увеличении показателей качества идентификации программ. Противоречие, возникающее между *недостаточной точностью* идентификации ПО существующими методами и *необходимостью* по обеспечению достаточного уровня защищенности информации в информационной системе, обуславливает **актуальность данного исследования**.

Целью работы является увеличение точности идентификации установленного программного обеспечения за счет разработки и обоснования научно-методического аппарата по идентификации исполняемых файлов, устанавливаемых на средства вычислительной техники, обеспечивающего увеличение точности идентификации в условиях наличия различных версий, большого числа различных наименований программ и ограниченности числа объектов обучающей выборки.

Для достижения поставленной цели были решены следующие **задачи**:

1. Исследование и анализ существующих подходов и методов идентификации программного обеспечения, используемых отечественными и зарубежными исследователями.
2. Разработка модели нарушителя информационной безопасности организации.
3. Разработка метода выделения признакового пространства исполняемых файлов с последующим созданием сигнатур на его основе.
4. Разработка метода сравнения сигнатуры идентифицируемого исполняемого файла с эталонными сигнатурами программ, обеспечивающего увеличение точности идентификации.
5. Разработка методики идентификации исполняемых файлов на основе созданного архива эталонных сигнатур программ с использованием статистического подхода и машинного обучения.

6. Проведение вычислительного эксперимента и обоснование применимости разработанной методики идентификации программного обеспечения.

В соответствии с заявленными целями и задачами работы: **объектом исследования** является разнообразие версий программного обеспечения, а **предметом исследования** методы идентификации программного обеспечения на основе статического анализа характеристик дизассемблированного кода программ.

В результате проведенных исследований были получены следующие результаты, составляющие **научную новизну** диссертации:

1. Метод формирования сигнатур исполняемых файлов, основанный на построении частотного распределения каждой из градаций выделенной характеристики исполняемых файлов, отличающийся от существующих использованием ряда отобранных наиболее информативных и устойчивых в проявлении ассемблерных команд.

2. Метод сравнения сигнатур идентифицируемых исполняемых файлов с ранее сформированными эталонными сигнатурами программ, отличающийся от известных применением комбинированного подхода использования алгоритма машинного обучения и аддитивного критерия, способствующего снижению числа ошибочных результатов классификации и обеспечивающего увеличение точности от совокупного использования признакового пространства, а также учитывающий ряд изменений в коде исполняемых файлов и позволяющий идентифицировать не рассматриваемые на этапе обучения версии программ.

3. Разработанная методика идентификации ПО, основанная на комбинированном анализе характеристик дизассемблированного кода программ, отличающаяся от известных, применением уникального сформированного признакового пространства и теории полезности для принятия решения на основе аддитивного критерия, что позволяет распознавать версии программ, ранее не задействованных в создании эталонных сигнатур исполняемых файлов

Теоретическую и практическую значимость работы составляют разработанные методы и методика по идентификации программного обеспечения,

позволяющие обнаруживать нарушения информационной безопасности при обработке конфиденциальной информации, вызванные несанкционированной установкой программ. Проведенные вычислительные эксперименты подтверждают результативность предложенных решений на реальных данных.

Методология и методы исследования поставленных задач включали теорию информационной безопасности, методы математической статистики, теорию предпочтений, методы машинного обучения, экспериментальные методы исследования.

На защиту выносятся следующие положения:

1. Метод формирования эталонных сигнатур программ и сигнатур идентифицируемых исполняемых файлов, основанный на статическом подходе анализа характеристик дизассемблированных кодов программ, обеспечивающий создание уникальных по форме и амплитуде частотных распределений, построенных на использовании ряда отобранных наиболее информативных ассемблерных команд, устойчиво проявляющихся в различных программах.

2. Метод сравнения сигнатур идентифицируемых исполняемых файлов с ранее сформированными эталонными сигнатурами программ при помощи комбинированного подхода использования алгоритма машинного обучения – градиентного бустинга деревьев решений и аддитивного критерия Фишберна, позволяющий достигать наименьшего числа ошибок неверной классификации и максимизировать эмерджентность совокупности признакового пространства.

3. Методика идентификации исполняемых файлов, включающая разработанные методы по формированию и сравнению сигнатур идентифицируемых исполняемых файлов с эталонными сигнатурами программ из архива эталонных сигнатур программ, обеспечивающая увеличение точности идентификации при комбинированном анализе характеристик из их дизассемблированного представления.

Обоснованность и достоверность полученных результатов подтверждается использованием апробированного математического аппарата; проведением сравнительного анализа с существующими методами; серией практических

экспериментов по идентификации исполняемых файлов; проверкой адекватности положений и выводов; согласованностью результатов, полученных при теоретическом исследовании с результатами проведенных экспериментов; практической апробацией результатов исследования в докладах и публикациях на отечественных и зарубежных научных конференциях.

Результаты практической **апробации работы** подтверждают адекватность и корректность разработанных методов. Основные результаты работы были представлены на следующих конференциях: 18th, 20th Conference of Open Innovations Association FRUCT and ISPIT 2017 seminar, 2016, 2017гг.; IV, VI, VII, VIII Всероссийский конгресс молодых ученых, 2015, 2017, 2018, 2019гг.; 11th IEEE International Conference on Application of Information and Communication Technologies, AICT 2017, 2017г.; IX, X Санкт-Петербургская межрегиональная конференция «Информационная безопасность регионов России (ИБРР-2017)», 2015, 2017гг.; Региональная информатика "РИ-2016", 2016г.; XLVII, XLVIII Научная и учебно-методическая конференция Университета ИТМО, 2018, 2019гг.; International Conference on Next Generation Wired/Wireless Networking Conference on Internet of Things and Smart Spaces NEW2AN 2018, ruSMART, 2018г.; 28-я научно-техническая конференция. Методы и технические средства обеспечения безопасности информации, МиТСОБИ, 2019г.

Результаты, полученные в диссертации, были реализованы в рамках выполнения следующих НИР: Проект по программе Президиума РАН № 0073-2018-0008 «Теория и распределенные алгоритмы самоорганизации группового поведения агентов в автономной миссии», 2018,2019гг.; Проект по программе Президиума РАН № 0073-2018-0007 «Разработка масштабируемых устойчивых алгоритмов построения семантических моделей больших данных и их использование для решения прикладных задач кластеризации и машинного обучения», 2018,2019гг.; НИР-ФУНД «Разработка методов интеллектуального управления киберфизическими системами с использованием квантовых технологий» №617026 (2017-2018гг.); НИР-ФУНД «Разработка методов создания и внедрения киберфизических систем» № 619296 (2018-2019гг.). Результаты

работы использовались при разработке системы мониторинга состояния внутренних сетей компании АО «НПК «ТРИСТАН». Полученные результаты используются в образовательном процессе факультета БИТ Университета ИТМО по направлениям подготовки бакалавриата 10.03.01 и магистратуры 10.04.01 по дисциплинам «Организация и управление службой защиты информации», «Теория вероятностей», «Методы цифровой обработки видеоизображений», «Управление информационной безопасностью».

Публикации. По результатам диссертационного исследования автором опубликовано 32 работы, из них статей в журналах, рекомендованных ВАК РФ – 8 входящих в базы цитирования Web of Science и Scopus – 8, свидетельств о государственной регистрации программы для ЭВМ – 6, в прочих изданиях – 10.

Личный вклад автора. Результаты диссертационной работы получены автором самостоятельно. Автором проведен анализ существующих методов идентификации программного обеспечения. Проанализированы условия и ограничения применения каждого из методов.

Структура и объем диссертации. Диссертационная работа содержит введение, 4 раздела, заключение, список литературы и 6 приложений. Объем работы составляет 163 страницы. Работа включает 26 рисунков, 17 таблиц. Список литературы содержит 101 наименование.

Краткое содержание работы.

Во введении обоснован выбор темы настоящей работы и ее актуальность, представлена степень разработанности темы, определены объект, предмет и цель исследования; описаны частные задачи, обоснована теоретическая и практическая значимость получаемых результатов; раскрыты принципы используемых подходов и разработанной методики; сформулированы положения, выносимые на защиту и проведена апробация результатов исследования.

В первой главе представлен анализ существующей проблематики современного подхода к идентификации программного обеспечения. На основе выделения объекта защиты, цели реализации угроз, потенциальных уязвимостей и видов ущерба разработаны модель угроз и нарушителя информационной

безопасности при обработке информации конфиденциального характера в информационной системе.

Во второй главе произведен обзор современных решений в области идентификации исполняемых файлов, представленных в отечественной и зарубежной литературе. Проанализированы различные подходы к сбору характеристик файлов и методы сравнения наборов данных характеристик для нескольких файлов. Выявлены достоинства и недостатки данных методов. Сформулирована постановка задачи обеспечения безопасности АС от потенциальных угроз со стороны нелегитимно установленного ПО.

В третьей главе произведен анализ структуры и характеристик ELF-файла, его дизассемблирование и представление исходного кода на низкоуровневом языке ассемблера. Описаны подход к представлению ПО и модель представления исполняемого файла в настоящем исследовании. Произведено выделение признакового пространства и описано его дальнейшее использование в различных методах формирования сигнатур.

Представлено подробное описание разработанных методов формирования сигнатур – эталонных и идентифицируемых. Сформулированы методы сравнения сигнатур, основанные как на статистическом подходе, так и на машинном обучении. Описана методика идентификации исполняемых файлов на основе статического анализа характеристик дизассемблированного кода программ, включающая в себя также и этап постобработки результатов сравнения сигнатур для увеличения точности идентификации исполняемых файлов. Представлены ограничения, накладываемые на методику и условия ее использования.

Сделан вывод о том, что необходимо экспериментальное подтверждение работоспособности представленной методики и входящих в нее методов, а также выделение наиболее результативных из них, путем сравнения достигаемых итогов точности идентификации. А также сравнение получаемых результатов с результатами методов отечественных и зарубежных исследователей.

В четвертой главе с целью проверки точности идентификации исполняемых файлов при помощи разработанной методики была проведена серия

экспериментов, направленных на каждый разработанный метод сравнения сигнатур файлов в отдельности. Полученные результаты прошли сравнение с результатами других исследователей и на данном основании сделан вывод о том, что в условиях наличия множества классов и ограниченности объема обучающей выборки, разработанная методика обеспечивает более высокую точность идентификации.

На практике методика идентификации программного обеспечения может быть применена для повышения безопасности информационной системы организации, путем внедрения ее в качестве технической меры по аудиту программного обеспечения на электронных носителях информации, она позволяет осуществлять автоматизированную идентификацию ELF-файлов в соответствии с формируемыми архивами легитимных или нелегитимных программ. Выделен ряд смежных задач, решаемых разработанной методикой.

В заключении приведены основные результаты и выводы по разработанной методике идентификации исполняемых файлов на основе статического анализа характеристик дизассемблированного кода программ.

Глава 1. Постановка задачи обеспечения безопасности автоматизированной системы от потенциальных угроз со стороны несанкционированно установленного программного обеспечения

Под идентификацией программного обеспечения понимается процедура построения некоторой информативной модели (модель в виде математического кортежа) программы (сигнатуры) по выбранному признаковому пространству (ассемблерным командам), характеристики которой позволяли бы, с заданной точностью, найти соответствие между рассматриваемой (идентифицируемой) программой и, предопределенной ранее на этапе формирования архива сигнатур, конкретной программной. Другими словами, идентифицировать исполняемый файл означает распознать его как ту или иную программу. Под программным обеспечением рассматриваются исполнимые и компонуемые файлы 32х и 64х разрядностей формата ELF [1] в Linux операционных системах для архитектур процессоров x86 и x86-64.

1.1 Обзор проблематики современных подходов к идентификации программного обеспечения

Стремительный прогресс информационных технологий, их повсеместное внедрение во все сферы человеческой деятельности приводит к тому, что современная организация бизнес процессов становится полностью зависима от надлежащего функционирования информационных систем (ИС). Область, отвечающая за безопасность информации обрабатываемой в таких системах и ее ресурсов, является областью информационной безопасности (ИБ).

Понятие ИБ трактуется различными ГОСТами по-разному, может определяться как процесс: «защита конфиденциальности, целостности и доступности информации; кроме того, сюда могут быть отнесены и другие свойства, например аутентичность, подотчетность, неотказуемость и надежность» [2], как свойство: «свойство информации сохранять

конфиденциальность, целостность и доступность» [3], как состояние: «состояние защищенности информации [данных], при котором обеспечены ее [их] конфиденциальность, доступность и целостность» [4]. Однако, из всех определений вытекает главная цель информационной безопасности – обеспечение трех наиболее важных свойств защищаемой информации: конфиденциальности, целостности и доступности.

Основным документом, описывающим порядок и принципы обеспечения информационной безопасности в организации, является политика ИБ, определение которой вытекает из двух понятий ГОСТа Р ИСО/МЭК 27002-2012, а именно информационной безопасности «Защита конфиденциальности, целостности и доступности информации; кроме того, сюда могут быть отнесены и другие свойства, например, аутентичность, подотчетность, неотказуемость и надежность.» [2] и политики «Общее намерение и направление, официально выраженное руководством». Таким образом, политика информационной безопасности является совокупностью документированных управленческих решений, направленных на защиту необходимых свойств информации и ассоциированных с ней ресурсов.

Дополнительно необходимо отметить, что представляет собой понятие конфиденциальной информации, ведь по законам Российской Федерации, защите [5] подлежит секретная [6] и конфиденциальная информация [7]. Ранее понятие конфиденциальной информации давалось в ныне утратившем силу ФЗ от 20 февраля 1995 г. № 24-ФЗ: «документированная информация, доступ к которой ограничивается в соответствии с законодательством Российской Федерации» [8], теперь же в соответствии с ФЗ от 27 июля 2006 г. № 149-ФЗ понятие конфиденциальной информации вытекает из двух определений: конфиденциальность информации «обязательное для выполнения лицом, получившим доступ к определенной информации, требование не передавать такую информацию третьим лицам без согласия ее обладателя» и информация «сведения (сообщения, данные) независимо от формы их представления» [5]. Таким образом, конфиденциальная информация – это сведения, независимо от формы их

представления, которые не могут быть переданы лицом, получившим доступ к данным сведениям, третьим лицам без согласия их правообладателя.

При взаимодействии пользователя с информационной системой организации, ее ресурсами и обрабатываемой внутри информацией наступает потребность в регулировании такого взаимодействия, реализуемое за счет политики ИБ, мер и средств по обеспечению безопасности [9]. В настоящей работе идентификация ПО рассматривается как техническая мера обеспечения безопасности, подкрепляющая собой организационную меру, часто выраженную в положении установленной политики безопасности организации о запрете на несанкционированное установление ПО.

С каждым годом все более доступным становится доступ к различным страницам, сайтам сети Интернет, ориентированным на информирование интернет-пользователей о существующих уязвимостях операционных систем, программного обеспечения и эксплуатации нетривиальных способов обхода установленных на рабочих местах систем защиты информации. Так же в свою очередь не малую роль играет то, что Linux ОС и большое количество программного обеспечения для них, представляют собой свободное программное обеспечение, пользователи которого, в частности, имеют права на его свободное использование и, что самое важное, модификацию. Все вышеописанное ведет к тому, что аудит электронных носителей информации на предмет выявления несанкционированно установленного ПО становится важной и все более актуальной задачей. Разработанная методика позволяет выявить нарушения установленной политики безопасности при обработке конфиденциальной информации.

Действия пользователей АС, направленные против установленной политики безопасности в организации, способны стать источником роста числа уязвимостей системы и повлиять на ее ИБ. Причиной этого являются возможные дефекты ПО, наличие не декларированных возможностей, нелегальное использование интеллектуальной собственности, а также использование специальных программ, направленных на преодоление установленной защиты либо противоправных

действий внутри сети Интранет или Интернет. Последнее особенно актуально в области преступлений, связанных с компьютерной информацией [10].

Существующие подходы к идентификации установленного программного обеспечения обладают рядом недостатков и ограничений, обеспечивающие низкую точность идентификации версий не вредоносного ПО. Возникает задача повышения точности идентификации ПО.

1.2 Модель угроз информационной безопасности при обработке информации конфиденциального характера в информационной системе

В соответствии с описанными выше определениями ИБ, в обобщенном виде формулирующие поддержание наиболее важных свойств защищаемой информации: конфиденциальности, целостности и доступности, приведем ниже описание объекта и целей реализации угроз относительно задачи данного исследования (рисунок 1).

По определению объектом защиты информации является «информация или носитель информации, или информационный процесс, которые необходимо защищать в соответствии с целью защиты информации» [4], таким образом для данного исследования ***объектом защиты*** являются сведения конфиденциального характера, обрабатываемые в автоматизированной системе, для которых необходимо предотвратить утечку, несанкционированные и непреднамеренные воздействия. Так же в качестве объекта может выступать сама автоматизированная система, ее ресурсы, электронные носители информации, и программное обеспечение, представляющее собой потенциальный источник уязвимостей системы [11].

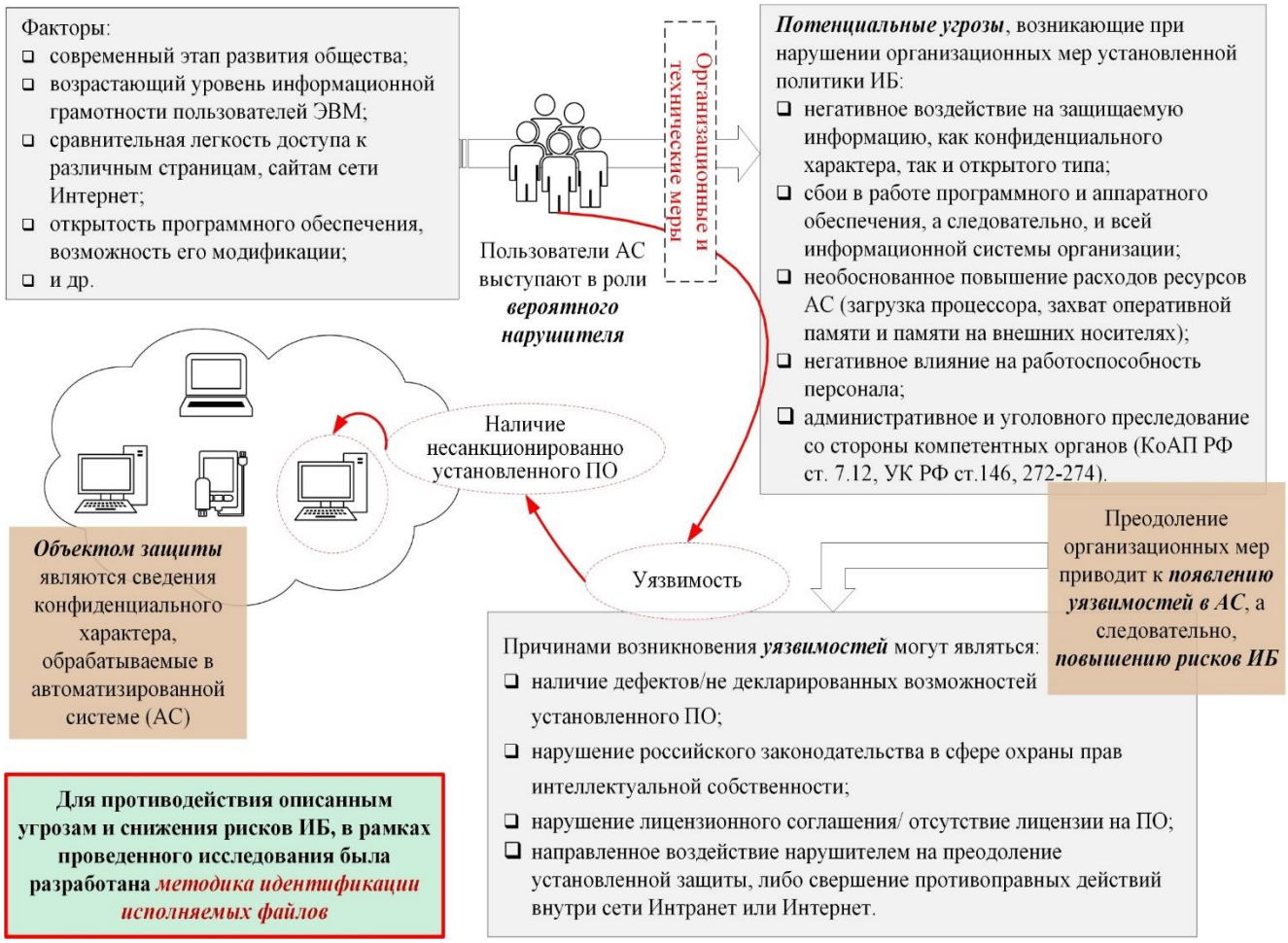


Рисунок 1 – Взаимосвязь факторов, угроз и появление новых уязвимостей при несанкционированной установке ПО

Целью реализации угроз является осуществление личных интересов пользователем АС, посредством преднамеренной установки несанкционированного ПО, способного привести к материальному или моральному ущербу организации.

В качестве ***потенциальных угроз***, способных привести к нижеописанному ущербу, выступают следующие угрозы, возникающие при нарушении организационных мер установленной политики ИБ организации:

– негативное воздействие на защищаемую информацию, как конфиденциального характера, так и открытого типа;

– сбои в работе программного и аппаратного обеспечения, а следовательно, и всей ИС организации;

– необоснованное повышение расходов ресурсов АС (загрузка процессора, захват оперативной памяти и памяти на внешних носителях);

– негативное влияние на работоспособность персонала;

– административное и уголовное преследование со стороны компетентных органов.

Преодоление организационных мер приводит к ***появлению уязвимостей в АС***, а следовательно, ***повышению рисков ИБ*** организации. Несанкционированно установленное ПО может не только стать лазейкой для противоправных действий нарушителя ИБ, но также привести к критическим ошибкам системы, способным остановить отлаженные бизнес-процессы, замедлить работу АС, повлиять на работоспособность персонала, путем переноса внимания работника от непосредственных обязанностей, деморализации коллективных отношений и т.д. Так использование нелицензионного ПО нарушает законодательство Российской Федерации и влечет наложение административной (если стоимость контрафактных экземпляров программ ЭВМ составит менее 50’000 рублей) или уголовной ответственности (если стоимость контрафактных экземпляров программ ЭВМ составит более 50’000 рублей) за нарушение авторских и смежных прав, которая предусматривается статьей 146 Уголовного кодекса Российской Федерации (УК РФ). При этом ответственность за противоправное использование ПО, в основном, возлагается на руководителя организации, принявшего управленческое решение по его использованию или же физическое лицо, самовольно установившее такое ПО. Отдельно необходимо отметить, что преступления по статье 146 УК РФ относятся к преступлениям публичного обвинения и не требуют подачи заявления с потерпевшей стороны. Ответственность же варьируется от административного штрафа в размере 1’500-2’000 рублей для физического лица до лишения свободы на срок до 6 лет со штрафом в размере до 500’000 рублей. Так, согласно исследованию BSA | The Software Alliance [12], было отмечено, что использование нелицензионного ПО в 2017 году составило 62%. При этом, коммерческая стоимость нелицензионного ПО, составила 1,291 миллиарда долларов США. По

данным отчета о видах наказания по наиболее тяжкому преступлению (без учета сложения) за 2017 год к уголовной ответственности по статье 146 УК РФ было привлечено 482 человека [13].

Причинами возникновения ***уязвимостей*** могут являться:

– наличие дефектов/не декларированных возможностей установленного ПО;

– нарушение российского законодательства в сфере охраны прав интеллектуальной собственности;

– нарушение лицензионного соглашения/ отсутствие лицензии на ПО;

– направленное воздействие нарушителем на преодоление установленной защиты, либо свершение противоправных действий внутри сети Интранет или Интернет.

В качестве ***потенциального ущерба*** от реализации угроз может быть причинен:

– моральный и материальный ущерб деловой репутации организации, возникающий в результате выявления надзорными органами нарушений использования чужой интеллектуальной собственности;

– моральный, физический или материальный ущерб, связанный с разглашением персональных данных отдельных лиц, который может возникнуть в результате использования ПО имеющего дефекты или направленного на преодоление установленной защиты;

– материальный ущерб от разглашения защищаемой информации, который может возникнуть в результате возникновения уязвимостей в АС при использовании ПО имеющего дефекты или направленного на преодоление установленной защиты;

– материальный ущерб от необходимости восстановления нарушенных защищаемых информационных ресурсов, доступ или целостность которых может быть нарушена;

– моральный и материальный ущерб от дезорганизации деятельности организации, вызванный уменьшением работоспособности пользователей АС;

– материальный и моральный ущерб от нарушения международных отношений, который может возникнуть в связи с нанесенным ущербом деловой репутации организации.

Источниками угроз являются пользователи АС, реализующие деятельность по преднамеренной несанкционированной установке ПО с целью извлечения своей выгоды. Умысел может носить разнообразный характер, как подстраивайте АС «под себя», так и, намеренная дестабилизация организованной системы защиты и добывание защищаемой информации.

1.3 Модель вероятного нарушителя информационной безопасности и оценка его возможностей

По признаку принадлежности к АС организации все нарушители будут являются внутренними, в силу того, что у них имеются некоторые права допуска в помещения и доступа к ЭВМ.

Естественно, по отношению к АС организации нарушителем может являться и внешнее лицо, но, если предполагать, что ИБ организации находится в устойчивом состоянии до того, как произойдет инцидент с несанкционированной установкой ПО, тогда можно считать, что для преодоления установленных средств защиты информации (ЗИ) внешнему нарушителю необходимы вначале действия внутреннего, которые повлекут за собой появление ранее не рассматривавшихся уязвимостей.

Возможности внутреннего нарушителя ограничены действующим в организации, в первую очередь, комплексом действующих в пределах контролируемой зоны режимных и организационно-технических мер, направленных, в рамках настоящей тематики работы, на пресечение действий пользователей АС по несанкционированной установке ПО.

Однако в зависимости от многих факторов (например, количеству денежных средств, выделяемых на организацию ИБ; наличию и уровню подготовки специалистов по ИБ; размерам штата и ресурсов в организации; добросовестности и уровню подготовки персонала в сфере ИБ; и т.п.), организационно-технические меры могут быть не достаточными и в таком случае у внутреннего нарушителя появляется возможность преодоления установленной политики безопасности.

Исходя из особенностей функционирования различных типов организаций, обрабатывающих информацию, подлежащую защите, к ***внутренним нарушителям*** можно отнести следующие общие категории персонала [14]:

– лица, имеющие непосредственный доступ к АС, но не имеющие доступа к защищаемой информации, при этом действия пользователей, относящихся к данной категории способны привести к дестабилизации комплекса защиты информации и увеличению рисков ИБ в сегменте локальной сети, где они расположены (категория 1);

– лица, имеющие непосредственный доступ к АС и имеющие доступ к защищаемой информации, а также удаленный доступ по локальным и распределенным информационным системам, при этом действия пользователей, относящихся к данной категории, в дополнение к вышеописанной первой категории, способны привести к нежелательному воздействию на свойства защищаемой информации (категория 2);

– лица, имеющие непосредственный доступ к АС и имеющие полномочия администратора безопасности сегмента информационной системы организации, здесь действия пользователя имеют более весомый характер воздействия на состояние защищенности информации, в дополнение руководящие позиции попадают под ответственность о защите интеллектуальной собственности (категория 3);

– лица, имеющие непосредственный доступ к АС и имеющие полномочия системного администратора или администратора безопасности информационной системы организации, здесь следует учитывать, что данная

категория пользователей обладает бóльшими правами и возможностями, знаниями относительно технических средств обработки информации, а также используемом программным обеспечением (категория 4);

– лица, занимающиеся разработкой, сопровождением и ремонтом технических средств в информационной системе организации, могут являться штатными/внештатными сотрудниками и производить несанкционированные действия по установке ПО (категория 5).

Стоит также отметить, что к защищаемой информации относятся не только сведения, попадающие под указ Президента РФ №188 [7], но и общедоступная информация, ограничение доступа к которой быть не должно [5], а также информация, дестабилизирующее воздействие на которую (уничтожение, модификация, искажение, копирование, блокировка), может привести к моральному или материальному ущербу организации и ее деловой репутации.

Предполагается, что пользователи всех категорий способны реализовывать угрозы ИБ, используя стандартное оборудование, расположенное в локальной сети организации, воздействие через которое способно повлиять на всестороннюю комплексность ИБ.

Кроме того, предполагается, что лица категории 3 и 4 хорошо знакомы с протоколами и алгоритмами, задействованными в информационной системе организации, и могут располагать специализированным оборудованием и программным обеспечением, позволяющим беспрепятственно производить несанкционированную установку ПО.

Для категорий 1-4 справедлива угроза снижения работоспособности персонала.

Предполагается, что для крупных организаций лица категории 3 и 4 проходят тщательную проверку со стороны руководства организации и являются доверенными. К сожалению, для более малых организаций такой тщательной проверки на квалифицированность и добропорядочность не производится, поэтому данные категории наравне с остальными могут относиться вероятным нарушителям.

Возможны следующие ***способы реализации угроз*** ИБ:

– несанкционированный доступ к защищаемой информации за счет использования функциональных особенностей несанкционированно установленного ПО, либо наличии в нем не декларированных возможностей, либо уязвимостей старых/новых версий;

– несанкционированный доступ к средствам обеспечения защиты информации;

– негативное воздействие на программно-технические компоненты ИС;

– использование программ игрового, информационного, развлекательного характера или предназначенная для обмена разного вида сообщениями;

– отправка сведений о нелицензионном использовании ПО компетентным органам или не прохождение их аудита.

Таким образом, по причине существующих недостатков современных средств по проверке программного обеспечения, возможностей нарушителя и возможных убытков, необходимо производить более тщательный и своевременный аудит электронных носителей информации на наличие несанкционированно установленного ПО.

Для противодействия описанным угрозам и снижения рисков ИБ, в рамках проведенного исследования были разработаны методы идентификации версий ПО.

Выводы по главе 1

а) В первой главе поставлена задача обеспечения безопасности автоматизированной системы от потенциальных угроз со стороны несанкционированно установленного программного обеспечения и конкретизирован вид рассматриваемого программного обеспечения.

б) Произведён анализ проблематики современных подходов к идентификации ПО, обозначена актуальность исследований в области информационной безопасности, описаны основные нормативные документы в области защиты информации и использования ПО.

в) Разработана и представлена модель угроз информационной безопасности при обработке информации конфиденциального характера в информационной системе, где также приводится классификация причин возникновения уязвимостей в автоматизированной системе при несанкционированной установке ПО и потенциальный ущерб организации.

г) Разработана и представлена модель вероятного нарушителя информационной безопасности и оценка его возможностей, позволяющая сделать вывод о том, нарушителем может быть пользователь как первой категории, так и пятой.

Глава 2. Обзор существующих методов идентификации программного обеспечения

В задаче идентификации объект $o \in O$ (где O – множество объектов) описывается при помощи некоторого выделенного признакового пространства $f_1, f_2,\ldots, f_n$, (где $i = 1, 2,\ldots, n$ – число градаций признака и значение для $f_n \in D_f$ – множеству допустимых значений признака) способного охарактеризовать основное свойство объекта $p(o)$, т.е. его класс, и записывается в виде вектора значений признака $o = (f_1(o), f_2(o) ,\ldots, f_n(o))$, именуемое в дальнейшем сигнатурой программы. Предполагается, что существует функциональная связь между признаками $f_1, f_2,\ldots, f_n$ и основным свойством объекта $p(o)$, позволяющая определить значение свойства $p(o)$, в данном исследовании в качестве основного свойства выступает имя программы, по ранее составленной информации о других объектах $o_1,o_2,\ldots,o_n$ (называемых обучающей выборкой), и их свойств $p(o_1), p(o_2) ,\ldots, p(o_n)$.

Идентификация происходит путем применения выбранного алгоритма сравнения двух векторов значений признака объектов: o – идентифицируемого и o' – эталонного.

Различают следующие типы признаков [15]:

– количественные признаки – признаки, измеренные в определенных метрических (числовых) шкалах;

– качественные – признаки, измеренные в не метрических шкалах и используемые для выражения терминов и понятий, не имеющих цифровых значений.

Также в зависимости от множества допустимых значений признака, выделяют следующие типы [16]:

– бинарный – принимающий значение нуля или единицы, $D_f = \{0,1\}$;

– номинальный – где множество допустимых значений признака конечно, $|D_f| = const$;

– порядковый – где множество допустимых значений признака конечно и упорядочено, $|D_f| = const$ и $D_f = \{d_f \mid d_{fj} \leq d_{fj+1}, j = 1,\ldots, const\text{-}1\}$;

– количественный – где в качестве допустимых значений признака выступает множество действительных чисел $D_f \in \mathbb{R}$.

В свою очередь представление признаков может носить разнообразный характер, в частности являться перечислением признаков, векторным описанием файла, или же иметь графическое представление, записанное в древовидной или графовой модели. В настоящем исследовании применяются номинальные (номинативные) признаки – это признаки, измеренные в не метрической шкале наименований, представляющие собой набор числовых характеристик признака.

На сегодняшний день существует ряд подходов к идентификации программного обеспечения на электронных носителях информации. Процесс распознавания (или идентификации) программы можно разделить на два этапа: первый - выделение признакового пространства и второй - применяемый метод распознавания программы.

Подходы к распознаванию программ могут основаны как на структурной характеристике файла, описывающей его содержимое, так и на семантической составляющей, характеризующей смысл анализируемого кода программы. К последним относится интеллектуальная обработка данных в которую входят [17]:

– формальные методы – оперирующие фиксированной моделью объекта и использующие предопределенные алгоритмы обработки;

– эвристические методы – способные, в некоторой степени, воспроизвести процесс познания человека; они направлены на выявление неизвестных объектов.

Методы интеллектуальной обработки данных, как правило, включают проверку «корректности» исполняемых файлов, вычисление статистического распределения инструкций процессора или энтропии кода, а также выявление последовательностей инструкций, характерных для вредоносных программ. В связи с чем, использование данной группы методов в целях идентификации не вредоносного программного обеспечения не представляется возможным.

Ниже в таблице 1 Таблица 1 – представлены наиболее распространенные подходы в задаче идентификации исполняемого файла.

Таблица 1 – Подходы, используемые для идентификации программ

Способы сбора характеристик программы	Методы распознавания программ
– По статическим характеристиками рассматриваемого объекта – По динамическим характеристиками рассматриваемого объекта – Задействующие встроенные функции операционной системы или функциональность внедряемого программного агента	– Задание метрики схожести – Статистический анализ – Машинное обучение – Эвристический анализ

2.1 Методы идентификации, основанные на статическом характере анализа характеристик программного обеспечения

Статические методы [18,19] оперируют двоичным образом программы, хранящемся на электронном носителе информации, будь то внешний жесткий диск или оперативная память компьютера. Данный метод подразумевает под собой работу с исполняемым файлом или его дампом памяти как с массивом байтовой последовательности.

Статические методы, по способу сбора характеристик программы, являются наиболее распространенными. Данный класс методов подразделяется на несигнатурные (методы проверки целостности): сравнение с полной копией данных, сравнение контрольных сумм, контроль CRC (Cyclic Redundancy Check), хеширование, имитовставка, цифровая подпись, и сигнатурные (методы сравнения признаковых последовательностей): выделение «магического числа», сравнение характерных последовательностей байтов и др.

В средствах защиты информации активно используется сигнатурный (синтаксический) метод, заключающийся в сравнении байтовых последовательностей с эталонной сигнатурой.

Идентифицировать файл можно с разной степени конкретности, так самой простой идентификацией будет являться определение расширения файла по его сигнатуре, известной также как «магическое число», представляемое в виде строго определенного набора байт, как правило, внесенных в начало тела файла. Так, например, для ELF файла эта последовательность задана следующим набором байт – 7F 45 4C 46; а для EXE файлов – 4D 5A.

Несигнатурные методы также популярны в настоящее время, но сфера их применения ограничена и ниже будут описаны их достоинства и недостатки применительно к задаче исследования.

Одним из первых и наиболее простых методов обеспечения целостности файла разработка метода контрольной суммы. Первоначально под контрольной суммой понималось некоторое число, занимающее позицию младшего разряда байта, и вычисляемое при помощи сложения остальных 7 бит. Сегодня под данным определением находится огромное число разнообразных алгоритмов, некоторые из которых имеют широкое применение, а некоторые используются только для специфических задач. Самыми известными и популярными алгоритмами являются: SHA-1, SHA-2, SHA-3, CRC и MD5, при этом стоит отметить, что последние два алгоритма применяются все реже, т.к. не способны обеспечить достаточного уровня стойкости к коллизиям.

Хеширование является простейшим подходом к сравнению файлов [20], [21]. При вычислении хеш-функции в качестве исходных данных принимается исходная двоичная последовательность файла, далее в зависимости от выбранного алгоритма она разбивается на блоки фиксированной длины с последующим вычислением значения хеш-функции.

Достоинством хеш-функции является ее простота, скорость выполнения операции и размер получаемого значения хеш-функции, который всегда имеет одну и туже длину. Однако, существенным недостатком в задаче сравнения двух

файлов является то, что вычисляемое значение хеш-функции существенно зависит от модификации произведенной над файлом, при этом оно ни в коей мере не зависит от степени произведенного вмешательства над оригинальным файлом. Т.е. если сравнивать значение хеш-функции оригинального файла со значением хеш-функции оригинального файла, но с измененным хотя бы одним битом – эти значения будут совершенно различны.

Чтобы снизить влияние модификаций и сравнивать похожие файлы, был предложен подход с использованием контекстно-зависимого кусочного хеширования (Context Triggered Piecewise Hashing), который использует плавающее окно с фиксированным значением шага, таким образом, контекстно-зависимое кусочное хеширование представляет собой конкатенацию значений хеш-функций блоков файла. Данный подход использовался в работах [22], [23], [24]. Модификации алгоритма выбора длины окна, шага и границ образуемых блоков приводят к существенному улучшению при идентификации схожих файлов, однако остается ограничение на количество возможных вносимых изменений в оригинальный файл, показанное в приведенных статьях.

Цифровая подпись в исполняемых файлах используется для верификации программы ее автором, позволяя идентифицировать издателя файла (подлинность файла) и определить, не подвергался ли он изменениям (целостность файла), тем самым гарантируется безопасность программы для АС. Таким образом, если файл содержит некорректную цифровую подпись (или ее нет совсем), то это может означать, что данный файл опубликован ненадежным издателем или был изменен [25], однако существуют способы обхода проверки цифровой подписи и запуска недостоверного программного обеспечения [26].

Решение, представленное в статьях [27], [28] основано на предположении, что разные части одного и того же вредоносного обеспечения имеют схожий порядок кода и областей данных. Каждая такая область файла может характеризоваться не только ее длиной, но также и своей схожестью. Другими словами, файл может быть охарактеризован комплексностью его областей кода.

Данный подход состоит в использовании вейвлет-анализа для сегментирования файлов [29] с различными уровнями энтропии и последующим использованием редакционного расстояния между сегментами последовательности для определения степени сходства файлов. Предлагаемое решение имеет ряд преимуществ, помогающих эффективно обнаруживать вредоносные программы на персональных компьютерах. Данный подход не принимает во внимание функциональность анализируемых файлов и основан исключительно на определении сходства в коде программы и данных о позиции области, что делает алгоритм эффективным против множества способов защиты исполняемого кода, с другой стороны, такое сравнение приводит к ложным срабатываниям. Стоит отметить, что авторы статей рассматривали своей целью распознавание вредоносного ПО, в частности выявление упакованных программ, ориентируясь на характерные признаки их семейства.

В работах [30], [31], также описывается метод идентификации типа файла на основе энтропийного подхода построения статистического профиля файла. Однако, авторы производят разбиения бинарного кода программы на блоки переменной длины при помощи метода скользящего окна и с учетом границ, выявленных по статистическим свойствам. Выбор длины блока основывается на правиле о том, что каждый блок должен представлять собой статистически однородный набор данных произвольного размера. Степень подобия двух файлов вычисляется с помощью описанного в работе показателя степени подобия файлов и задействует редакционное расстояние пары блоков статистического профиля файлов.

Еще один подход по выделению признакового пространства был предложен в работе [32], где авторы представляют исходный файл в виде последовательности цифровых изображений с фиксированной шириной и высотой, определяемой на этапе выбора длины скользящего окна для алгоритма подготовки данных. Также авторами предлагается второй подход к формированию входных данных, основанный на энтропии различных участков кода исполняемого файла. Далее яркость каждого пикселя получаемого изображения принимается равной значению

конкретного байта из идентифицируемого объекта. Таким образом, исходный файл представляется в виде набора изображений с фиксированной шириной и высотой. По причине, возникающей нежелательной, но сильной зависимости получаемых изображений от длины скользящего окна и наличию смещений в идентифицируемых файлах относительно друг друга, авторы обращаются к такому методу классификации объектов как машинное обучение, а именно к неокогнитрону – иерархической многослойной нейронной сети, предназначенной для инвариантного распознавания образов. Однако сами авторы говорят о существенных ограничениях и недостаточной эффективности такого подхода для целей распознавания всех возможных семейств вредоносных файлов.

В работе [33] автор рассматривает идентификацию программ по внутренним характеристикам, при этом выделяет важность использования такого критерия установления степени подобия исходной и исследуемой программы, который был бы независимым от маскировок, вносимых в исходный текст программ нарушителем. Так, ссылаясь на работу [34], автор принимает решение об использовании операторов по тексту программ в качестве используемого признакового пространства. Повторяемость встречаемых операторов в коде программы делает степень подобия программ визуально видимой при построении их гистограмм, что особенно видно для программ имеющий схожее функциональное назначение, поскольку выбор операторов часто определяется целевой направленностью программы. В качестве меры оценки подобия в работе рассматривается корреляционная функция и поиск ее максимума между целевой и оцениваемой программами. Так наиболее схожие файлы будут иметь значение коэффициента корреляции близкое к единице, в свою очередь наименее схожие программы будут иметь значение, приближенное к нулю. Недостатком описанного подхода является то, что частота наиболее популярных операторов программы вносит существенный вклад в значение коэффициента корреляции и увеличивает его, что требует постоянного мониторинга и внесения изменений в пороговое значение коэффициента. Данный подход рассматривался автором в качестве меры по обеспечению безопасности программ, в случае, когда их исходные тексты

попадают в руки злоумышленников, желающих внести в код программы разрушающие программные средства до того, как программы подвергнутся компиляции.

В работе [35] для извлечения признаковых характеристик программ использовались три способа: составление бинарных профилей, последовательность строк и шестнадцатеричные дампы. Автор применял описанные методы к исполняемым файлам ОС Windows и, в качестве методов распознавания программ, применял алгоритмы машинного обучения, такой как наивный классификатор Байеса. Первый способ извлечения признаковой характеристики заключался в выделении трех типов информации о ресурсе из исполняемых файлов PE формата: списка динамически подключаемых библиотек DLL; функции, вызываемые этими библиотеками; числом отличных системных вызовов каждой из библиотек – и дальнейшем формировании вектора бинарных характеристик. В качестве метода по распознаванию программ выступал алгоритм RIPPER [36]. Второй способ извлечения признаков заключался в использовании UNIX команды strings, которая показывает печатаемые строки в объекте или двоичном файле. Авторы формировали обучающую выборку, рассматривая строки как двоичные атрибуты, которые либо присутствовали в данном исполняемом файле, либо отсутствовали. Третий способ использовал утилиту hexdump, которая представляет исполняемый файл в виде набора шестнадцатеричных чисел. Также, как и для второго способа, автор использовал двухбайтовые последовательности в качестве бинарных атрибутов, которые либо присутствовали в данном исполняемом файле, либо отсутствовали.

В работе [37] авторы рассматривают подход к идентификации вредоносных исполняемых файлов PE формата на основе применения алгоритмов машинного обучения к формируемым *n*-граммам, представляющим собой последовательность из n элементов. Они используют утилиту hexdump для конвертации всех рассматриваемых исполняемых файлов. Далее формируют *n*-граммы, состоящие из четырехбайтовых последовательностей. Среди алгоритмов по классификации исполняемых файлов были использованы: метод k-ближайших соседей, наивный

Байесовский классификатор, метод опорных векторов (SVM), деревья решений. Также применительно к последним трем представленным алгоритмам использовался бустинг. Авторы решали задачу классификации с двумя классами, первый из которых являлся «исполняемый файл распознался как вредоносный», второй – «исполняемый файл распознался как безвредный». Авторами были достигнуты хорошие результаты, представленные в виде ROC и AUC, достигающие 99,6% покрытия площади под ROC (AUC) для доверительного интервала 0,05. Также авторы произвели классификацию вредоносных исполняемых файлов используя обобщенные три класса, характеризующих ключевые особенности вредоносного ПО: массовая почтовая рассылка (AUC = 0,8986), backdoor (AUC = 0,8704), вирус (AUC = 0,9114). К сожалению, не представляется возможности сравнить предоставленные результаты, так как результаты настоящего исследования используют мульти-классы.

В работе [38] представлено несколько методов дизассемблирования двоичных исполняемых файлов, помогающих сформировать представление исполнения двоичных исполняемых файлов и улучшить разбиение программы на «идиомы» (так называемые последовательности инструкций). Также стоит упомянуть здесь работу [39], в которой предложен метод, основанный на анализе CFG для обработки вредоносных программах с обфускацией. Позже, в [40] был улучшен метод путем включения семантических шаблонов вредоносных характеристик.

В работе [41] исследовалась способность ассемблерных команд обнаруживать вредоносные программы. Его исследование показало, что ассемблерные команды показывают значительные различия между вредоносным и легитимным ПО. А также то, что редко встречаемые команды несут большую долю предсказания, чем часто встречаемые команды.

В [42] изучен подход к идентификации ранее не известного вредоносного ПО с помощью машинного обучения. Исполняемые файлы рассматривались как непрерывная последовательность ассемблерных команд. Поскольку большинство распространенных ассемблерных команд, которые могут использоваться для

вредоносных целей, требуют более одного машинного кода операции, было предложено использовать последовательности ассемблерных команд вместо отдельных команд. Таким образом, происходит добавление синтаксической информации, что, по мнению авторов, улучшает способность разработанного ими подхода к идентификации блоков последовательности команд, описывающих вредоносное поведение исполняемых файлов. В качестве идентификатора программы выступало чередование групп (вектор) из двух значений: частота встречаемости последовательности из *n*-грамм ассемблерных команд в файле и присвоенный им вес. В эксперименте принимали участие последовательности ассемблерных команд размерностей $n = 1$ и $n = 2$.

Схожий подход рассматривался в работах [43–46], где исследовалась точность классификации исполняемых файлов при изменении размера *n*-граммы; различном сборе характеристик программы – одновременное использование статических и динамических характеристик; в сфере IoT устройств; определенных семейств вредоносного ПО и т.д.

В работе [47] представляется исследование по классификации программ на основе алгоритмов, по выявлению плагиата. Обнаружение плагиата – это процесс выявления того, что некоторые части исследуемой работы не являются оригинальными наработками автора произведения. Одним из наиболее распространенных применений таких программ является обнаружение плагиата в программах, разработанными студентами. В данной работе считают, что обнаружение плагиата программного обеспечения и кластеризация вредоносных программ связаны друг с другом в том, что они оба пытаются обнаружить некоторую степень сходства между двумя программами среди большого числа исследуемых программ. Однако, из-за уникальности образцов вредоносных программ по сравнению с программами в целом (например, при использовании привилегированных системных ресурсов) и от степени обфускации, обычно применяемой в отношении вредоносных программ в работе был выражен скептический настрой на эффективность рассматриваемых подходов.

Работа [48] рассматривает в качестве характеристики файлов – строковый константы, получаемые путем сканирования файла, чтения каждого знака один за другим и созданием списка всех получаемых строк. Длина строки определялась как набор символов. Чем короче длина строки, тем больше вероятность, что характеристика будет иметь усредненную значимость в исследуемой выборке файлов. Однако такая длина предоставляет больше возможностей. В работе отмечается, что для улучшения производительности и избегания переобучения при использовании методов машинного обучения, возникает необходимость в создании фильтра характеристик. В качестве метода идентификации были исследованы три подхода на основе классификатора Байеса: наивный (Naive Bayes), мульти-классификатор (Multi-naive Bayes) и частично-приращенный (Half Increment Bayes). Наилучший результат был достигнут при помощи использования последнего классификатора, однако временная сложность оценивается авторами как $T_H = O(S_2 \times k_1 + S_3 \times (k_1 + k_2) + ... + S_n \times (k_1 + k_2 + ... + k_r))$, где k_r – характеристики строковых констант, извлеченных из r-ой выборки, S_n – строковые константы, извлеченные из n-ой выборки.

Характеристикой исполняемого файла в работе [49] служит статистический профиль – вектор статистических описаний файла, полученный при движении скользящего окна от начала к концу файла с шагом в один байт, и расчета в каждом положении окна некоторой статистической величины. Ширина окна обычно подбирается экспериментально. Затем в исследовании применяется метод разбиения файла на блоки фиксированной длины и подсчета их длин, полученных кодированием набора байтов при помощи алгоритма Хаффмана. Также авторами рассматриваются графики энтропий для программы, упакованной с помощью различных упаковщиков.

2.2 Методы идентификации, основанные на динамическом характере сбора характеристик программного обеспечения

Динамический анализ основан на запуске анализируемой программы на исполнение. Отсутствие каких-либо предположений о ходе исполнения программы

и проверка её в процессе или сразу после исполнения является значимым преимуществом динамического анализа. Очевидным требованием, предъявляемым при реализации динамического анализа – его проведение должно наименьшим образом влиять на ход исполнения. При определённых условиях на детерминированность программы, динамический анализ позволяет избежать проблемы ложных срабатываний.

Большинство современных вредоносных программ использует методы запутывания (обфускации), в частности бинарные упаковщики, чтобы помешать статическому анализу двоичных файлов. Таким образом, динамический анализ таких вредоносных программ зачастую бывает намного эффективнее статического анализа. Мониторинг поведения программ в ходе их исполнения включает, например, сбор выполняемых программой бинарных операций и предлагает потенциально более глубокое понимание самого кода. Пока подходы такого типа имеют свои ограничения, например – может быть трудно вызвать определенное поведение программ, а некоторые из них могут требовать определенные условия окружающей среды – тем не менее его использование считается более эффективным, по сравнению с только лишь статическими подходами. По этой причине, динамическому анализу программ уделяется большее внимание в научном сообществе. Системы анализа, как CWSandbox [50], Anubis [51], BitBlaze [52], AVG [53] и Advanced Threat Protection [54] выполняют запуск образцов вредоносных программ в защищенной среде [55] и отслеживают их поведение для анализа и разработки защитных механизмов.

Так основными считаются подходы по [56]:

– выявлению последовательности необычных путей исполнения кода программы или вызываемым системным функциям (API, system calls, Windows native API) [57];

– нанесенным повреждениям окружающей среды в изолированной системе [58];

– попыткам получения высоких привилегий [59];

- изменениям параметров вызываемых функций [35];
- способам обработки данных программой [60];
- трассировке машинных инструкций программы [61].

Данный подход является более эффективным в плане анализа программы, однако для задачи идентификации большого количества версий не вредоносных программ его применение становится не целесообразным, поскольку главный минус динамического анализа связан с большими временными затратами, требуемыми на выполнение и тщательное исследование программы.

2.3 Методы идентификации, основанные на сборе информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента

В настоящее время множество компаний обращаются к использованию систем управления ИТ-активами (ITAM), представляющие собой комплексные решения, нацеленные на физический учёт, финансовый контроль и соблюдение контрактных обязательств, связанных с ИТ-активами, на протяжении всего их жизненного цикла [62,63]. Здесь под ИТ-активами подразумеваются все аппаратные и программные элементы ИТ-инфраструктуры, обеспечивающие деятельность бизнес-среды.

В свою очередь ITAM подразделяется на управление аппаратными активами (HAM), охватывающее управление материальными составляющими ИТ-инфраструктуры: пользовательские компьютеры, сервера, телефоны и т.д.; и на управление программными активами (SAM), охватывающее управление не материальными составляющими ИТ-инфраструктуры: программное обеспечение, лицензии, версии, конечные точки инсталляции и т.д.

На рынке услуг представлено немало решений [64], позволяющих идентифицировать программные активы, управлять учетом, а также производить контроль их изменений и др. Из числа наиболее известных программных продуктов можно выделить:

– Работающие на основе агента:

1) Samanage – поддерживает работу с Windows, Linux, Mac ОС [65].

2) Kaspersky Systems Management – поддерживает работу с Windows, Linux, Android, iOS [66].

– Работающие без агента:

1) Microsoft Assessment and Planning Toolkit – поддерживает работу с Windows ОС [67].

– Работающие как с агентом, так и без него:

1) AIDA64 – поддерживает работу с Windows, Linux, Android [68].

2) Lansweeper – поддерживает работу с Windows, Linux, Mac ОС и устройствами, поддерживающими IP адресацию [69].

Агентом является программный агент, устанавливаемый на компьютер пользователя АС и ведущий скрытую деятельность по выполнению своих функций, в частности инвентаризации ПО. Недостатком данного подхода является необходимость установки агента на каждый исследуемый компьютер, однако преимуществом агента является возможность его установки на корпоративные ноутбуки и смартфоны, используемые пользователями за пределами внутренней сети организации, получают самый высокий уровень детализации, могут быть использованы для вмешательства в компьютер пользователя. Использование подхода без установки агента не требует наличия подходящих для данного агента систем, его обновлений, не повышает производительные затраты ресурсов компьютера, а также имеет удобное централизованное управление. Однако, уровень детализации получаемой информации ниже, чем при задействовании агента, а также необходимо удостовериться, что исследуемые активы доступны для их обнаружения.

Подход к оценке ресурсов с(без) агентом имеет свои преимущества и недостатки, особенно стоит учитывать число самих ресурсов в организации.

Однако стоит отметить, что большая часть предоставляемой информации собирается посредством встроенных технологий инвентаризации программного и аппаратного окружений, операционной системы:

– Windows Management Instrumentation – является одним из базовых инструментариев централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows;

– Active Directory Domain Services – служба каталогов Windows, которая централизованно сохраняет все данные и настройки среды в базе данных;

– SMS Provider – поставщик инструментария управления Windows, назначающий доступ на чтение и запись к базе данных Configuration Manager на сайте сервера;

– lshw – утилита Linux, которая выводит полный список аппаратных компонентов системы вместе с информацией об устройствах;

– dpkg -l – утилита Linux, которая выводит полный список программных компонентов системы;

– и других технологий.

Недостатки такого подхода очевидны. Отсутствие должного уровня оценивания роли информационной безопасности в бизнес процессах организации приводит к недооценке кадровой политики со стороны руководства при подборе ИТ персонала. В то же время, пользователи АС имеют возможность достичь достаточного уровня компетенции в сфере компьютерных технологий, позволяющие им обходить установленные политики безопасности. Все это предоставляет возможность внести изменения в конфигурационные данные устанавливаемого программного обеспечения.

Ниже на рисунках 2-4 приведены примеры по изменению версии программ (*ABBYY Lingvo* и *Htop*) в операционных системах Windows (10) и Linux (Ubuntu) соответственно. Из которых становятся очевидны пути обмана средств аудита программного обеспечения как для установленного ранее программного обеспечения, так и для устанавливаемого впервые.

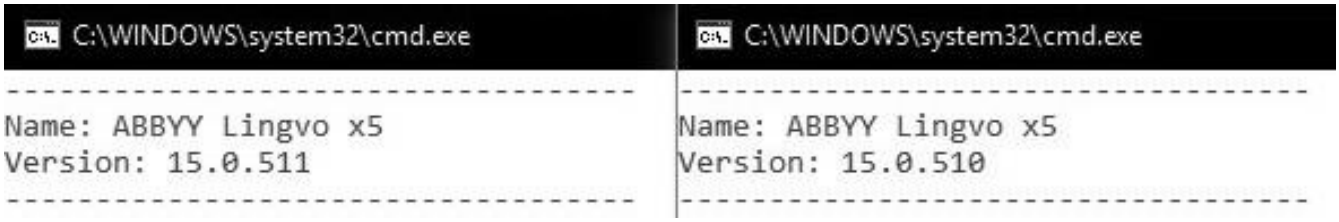


Рисунок 2 – Изменение версии программы ABBYY Lingvo x5 15.0.511 на ABBYY Lingvo x5 15.0.510

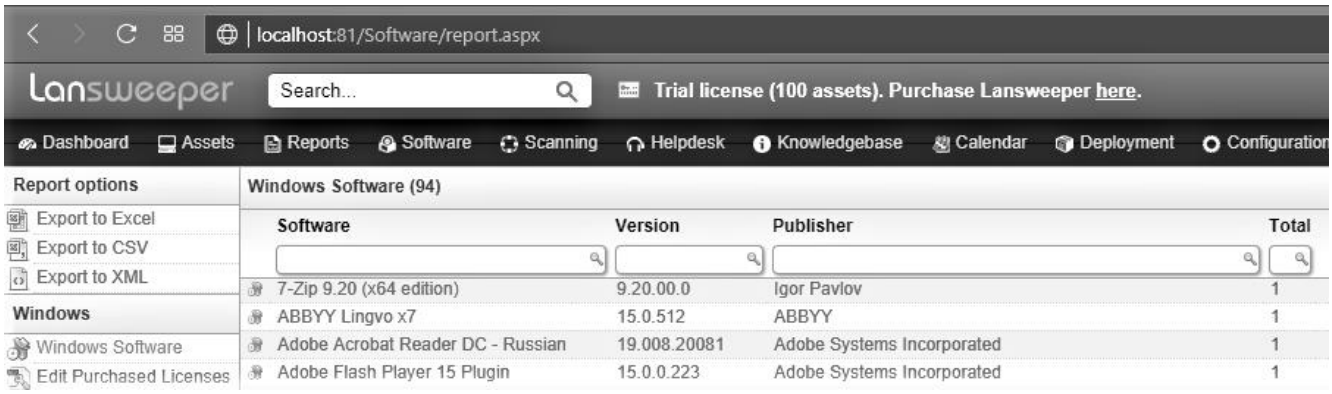


Рисунок 3 – Изменение версии программы ABBYY Lingvo x5 15.0.511 на ABBYY Lingvo x7 15.0.512

Из рисунков 2 и 3 видно, что путем произведенных манипуляций в реестре операционной системы Windows, версия программы *ABBYY Lingvo* была изменена дважды, целью данного вмешательства являлось намеренное влияние на результаты выдачи запроса по инвентаризации программного обеспечения с помощью средств Windows Management Instrumentation (WMI) (результат выдачи представлен на рисунке 2) и распространенного программного средства Lansweeper (результат выдачи представлен на рисунке 3).

На рисунке 4 в свою очередь представлена установка программы *Htop* в операционной системе Ubuntu с заведомо измененной версией, исправленной путем вмешательства в конфигурационный файл устанавливаемого пакета .deb.

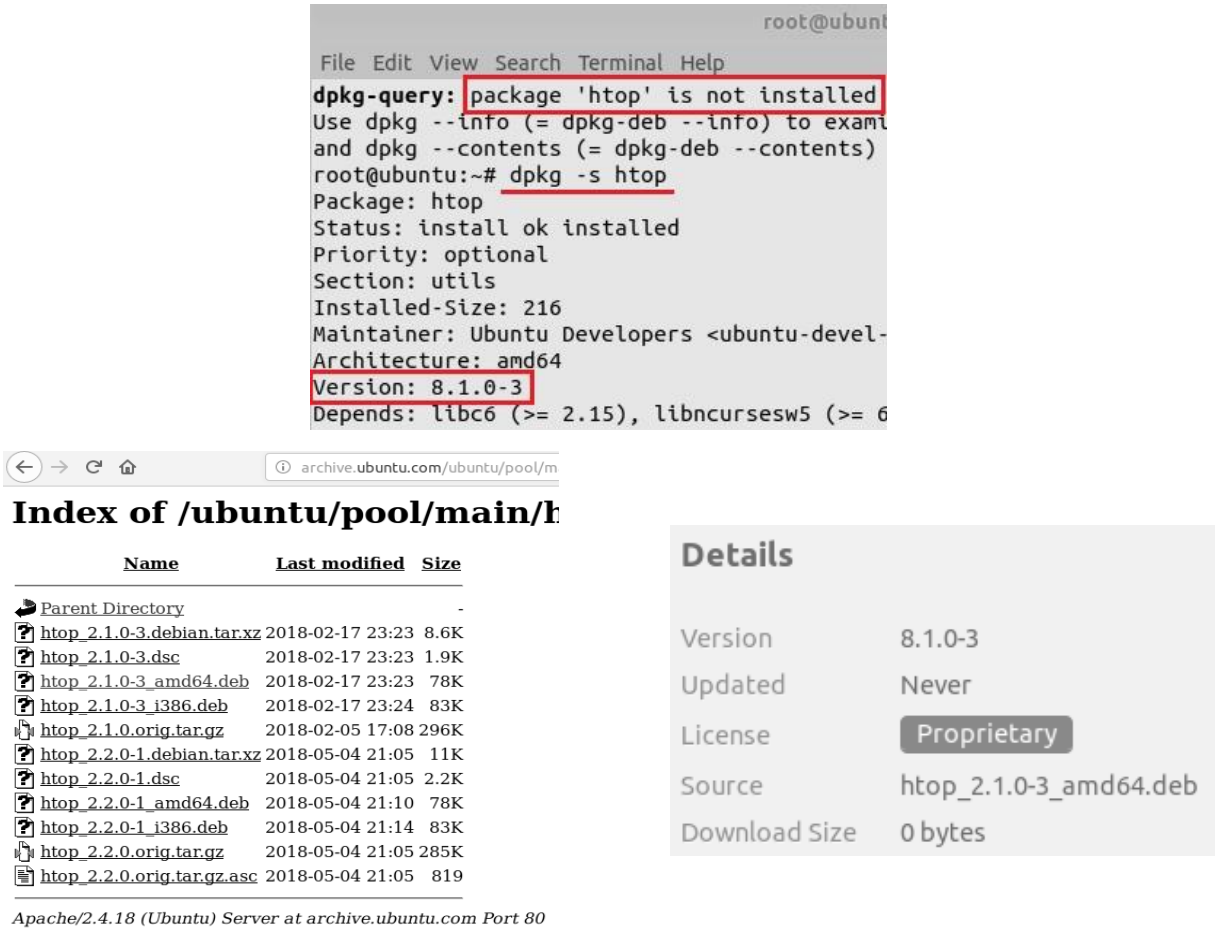


Рисунок 4 – Установка измененной версии программы htop 8.1.0-3 вместо htop 2.1.0-3

Сравнение наиболее близких к задаче исследования работ по идентификации исполняемых файлов представлено ниже в таблице 2.

Таблица 2 – Сравнение основных работ по идентификации исполняемых файлов

Источник	Формат файлов	Используемые характеристики файла	Метод идентификации	Число классов	Оценка качества метода[1]	
					Точность (Accuracy), %	Бикубическая мера качества кластеризации
1	2	3	4	5	6	7
Data mining methods for detection of new malicious executables, [35]	PE/?[2]	Шестнадцатеричный дамп	Signature	Бинарная классификация	49,31	-
		Используемые DLL	RIPPER		83,61	-
		функции, вызываемые DLL	RIPPER		89,36	-
		Число отличных системных вызовов каждой из DLL	RIPPER		89,07	-
		Печатаемые строки	Naive Bayes		97,11	-
		Набор шестнадцатеричных чисел	Voting Naive Bayes		96,88	-
Opcode sequences as representation of executables for data-mining-based unknown malware detection, [42]	PE	Последовательность частоты встречаемости очередности (n = 1) ассемблерных команд	SVM: Pearson VII		92,92	-
		Последовательность частоты встречаемости очередности (n = 2) ассемблерных команд	SVM: normalised polynomial		95,90	-
		Последовательность частоты встречаемости очередности (n = 1 и n = 2) ассемблерных команд			95,80	-

[1] Показана максимально достигаемая величина для используемом способе сбора характеристик файла и методе идентификации

[2] Только часть файлов формата PE, для остальной части файлов формат не известен.

Продолжение таблицы 2

1	2	3	4	5	6	7
On Challenges in Evaluating Malware Clustering, [47]	?	Последовательность частоты встречаемости очередности вызовов API (n = 3)	Мера Жаккара + кластеризация на основе работы [70]	Кластеризация	-	0,81
		Последовательность вызовов API	Сравнение строк программными средствами (diff, CCFinder) + кластеризация на основе работы [70]		-	0,78
Identifying almost identical files using context triggered piecewise hashing, [22]	PE	Побайтовый код программы	Контекстно-зависимое кусочное хеширование	Мульти-классификация	-	0,29
A Fast Randomness Test that Preserves Local Detail, [49]		Вектора для блоков постоянного размера побайтового кода программы	Евклидово расстояние		-	0,55
Идентификация типа файла на основе структурного анализа, [30]	?	Вектора для блоков постоянного размера побайтового кода программы	Редакционное расстояние		-	0,69
Identification of Executable Files on the basis of Statistical Criteria, [71]	ELF	Последовательность частот встречаемости 118 ассемблерных команд	Статистический критерий однородности хи-квадрат		98,67/ 79,55[3]	-
		Последовательность частоты встречаемости одной ассемблерной команды на неравных интервалах	Статистический критерий согласия Колмогорова		95,09/ 50,41[3]	-

[3] Оценивание точности статистического критерия происходило двумя способами (*Confusion matrix* и *Accuracy*), описание которых представлено в разделе 2.6.

Продолжение таблицы 2

1	2	3	4	5	6	7
Подход к выбору информативного признака в задаче идентификации программного обеспечения, [72]	ELF	Последовательность частоты встречаемости одной ассемблерной команды на равных интервалах	Статистический критерий однородности хи-квадрат	Мульти-классификация	98,58/ 78,05[3]	-
Разработка способа идентификации elf-файлов на основе нейронной сети, [73]			Нейронная сеть MLP, с методом сопряженных градиентов и 50 нейронами		82,11	-
Алгоритм градиентного бустинга деревьев решений в задаче идентификации программного обеспечения, [74]			CatBoost		89,43	0,88
Сравнительный анализ подходов к идентификации программного обеспечения, [75]			LightGBM		86,99	0,84
			XGBoost		95,93	0,96
Повышение точности идентификации программного обеспечения путем использования аддитивного критерия Фишберна, [76]			XGBoost и Фишберн		99,19	0,99

В данной главе был произведен обзор существующих методов идентификации программного обеспечения, из которого можно выделить ряд ограничений, не позволяющий их использование в задаче идентификации версий программного обеспечения:

– Для статического сбора характеристик файла:

1) Методы основанные на оценке целостности файла не позволяют исследовать схожесть программ, не задействованных в обучающей выборке.

2) Использование особенностей PE формата программ ОС Windows, отсутствующие в ELF-файлах.

3) Большинство существующих подходов к классификации исполняемых файлов направлены на бинарное разделение тестовой выборки и выделение класса вредоносных программ.

4) Так же часть подходов рассматривают мульти-классификацию с ограниченным набором классов.

5) Работы же, направленные на мульти-классификацию с неограниченным набором классов, обладают низкой точностью идентификации исполняемых файлов.

– Для динамического сбора характеристик файла:

1) Использование данного типа подходов является не целесообразным, по причине задействования огромного числа ресурсов.

2) Для выполнения программы бывает необходимо создавать особую среду исполнения.

– Для сбора информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента:

1) Главным недостатком является сам принцип получения информации с АС, где факт наличия способов изменения информации о программе был приведен в соответствующем пункте 1.4.3.

Автором научной работы были проанализированы методы идентификации исполняемых файлов АС, использующие в качестве идентификационных

признаков последовательности системных вызовов, байтов, ассемблерных команд, энтропийных характеристик и др. Представлены их ограничения в возможностях идентификации программ.

Приведенный анализ методов идентификации различных версий исполняемых файлов, позволяет сделать вывод, что идентификация на основе статического анализа характеристик программы и использование статистических и машинных алгоритмов классификации, является наиболее перспективным направлением деятельности в рамках поставленной научной задачи исследования.

Таким образом, в современных условиях, не существует решения, позволяющего с достаточной точностью производить идентификацию различных версий ПО в АС в задаче по выявлению факта нарушения пользователем установленных правил по несанкционированной установке ПО. Отсюда вытекает необходимость в проведении настоящего исследования, направленного на достижение максимальной точности идентификации исполняемых файлов, путем совершенствования существующих подходов, поиска новых, ранее не применявшихся, решений в области выделения признакового пространства файлов, методов идентификации, а также способов постобработки результатов.

2.4 Постановка задачи исследования

В качестве идентифицирующей исполняемые файлы информацией – сигнатуры, выступают структурные характеристики программы, считываемые с помощью подхода, основанном на статическом характере сбора. Имеются сигнатуры идентифицируемого исполняемого файла (СИИФ) и эталонные сигнатуры программ (ЭСП).

Идентифицировать исполняемый файл при аудите электронных носителей информации означает распознать программу путем анализа ее структурных характеристик и сравнения сформированной СИИФ с ЭСП некоторых известных нам потенциальных программ, хранимых в архиве, или сделать вывод о том, что программа не является ни одной из нам известных программ.

Определение того, является ли идентифицируемый файл той или иной программой (принадлежит определенному классу) из числа возможных потенциальных программ, производится на основании анализа значений выделенного признакового пространства СИИФ и ЭСП, хранимых в архиве сигнатур программ (АЭСП).

Архив может строиться на основе одного из принципов – в него заносятся сигнатуры программ, нахождение которых на электронном носителе информации *разрешено*; либо в него заносятся сигнатуры программ, нахождение которых на электронном носителе информации *запрещено*. Выбор одного из принципов зависит от поставленных целей и задач при аудите.

Исходя из описанного выше, задачу идентификации исполняемых файлов по статическим характеристикам можно разделить на следующие две подзадачи:

1) Формирование АЭСП A путем сбора выбранной характеристики F (feature) различных версий эталонных программ $P_{\text{этал.},i}$ (program), и создание на их основе ЭСП $S(P_{\text{этал.},i})$ (signature).

2) Идентификация неизвестного исполняемого файла путем сравнения его СИИФ $S(e_j)$ с ЭСП и последующей постобработкой результатов. В случае выдачи положительного результата применяемым методом идентификации, исполняемый файл e_j (executable) считается идентифицированным как та или иная программа $P_{\text{этал.},i}$, или же как не файл, не относящийся ни к одной из программ АЭСП.

Задача идентификации ПО в общем случае сводится к решению задачи мульти-классификации.

Для каждого идентифицируемого файла необходимо определить его меру близости с потенциальной эталонной программой из АЭСП, т.е. необходимо отнести исполняемый файл к определенному классу известных нам программ.

Имеется множество дизассемблированных представлений потенциальных эталонных программ 32-х и 64-х разрядностей формата ELF в Linux ОС для архитектур процессоров x86 и x86-64, взятых с официальных репозиториев и обработанных программой objdump $P = \{P_{\text{этал.},1}, P_{\text{этал.},2}, \ldots, P_{\text{этал.},i}\}$ и их имен $N = \{N_1, N_2, \ldots, N_i\}$, при этом функция соответствия программы $P_{\text{этал.},i}$ и имени

программы N_i – биективна, где i – число программ разных наименований формирующих АЭСП (обучающая выборка). Так же есть множество версий для программ $P_{\text{этал.},i} = \{v_1, v_2, \ldots, v_m\}$, где m – число версий программы с одним наименованием, которые участвуют в формировании сигнатур $S(P_{\text{этал.},i})$.

В свою очередь для формирования сигнатуры версии программы используется функция $q(v_m,F)$: $X \rightarrow N_0$, которая при помощи заданного алгоритма и выбранной характеристики F формирует частотную последовательность признака версии программы, где X – множество принимаемых значений характеристики F. Затем полученные сигнатуры версий программы формируют наборы ЭСП $S(P_{\text{этал.},i}) = \{(N_i, q(v_1,F)), (N_i, q(v_2,F)), \ldots, (N_i, q(v_m,F))\}$, где N_i – имя программы $P_{\text{этал.},i}$, которые заносятся в АЭСП: $A = \{S(P_{\text{этал.},1}), S(P_{\text{этал.},2}), \ldots, S(P_{\text{этал.},i})\}$.

При проведении аудита имеется некоторое множество неизвестных исполняемых файлов $E = \{e_1, e_2, \ldots, e_j\}$ (тестовая выборка), где j – число исследуемых исполняемых файлов. С помощью той же выбранной характеристики F и функции q производится формирование частотной последовательности признака для каждого исследуемого файла e_j, СИИФ $S(e_j)$ представляет собой результат функции $q(e_j,F)$: $X \rightarrow N_0$.

Требуется построить алгоритм $I(S(P_{\text{этал.},i}), S(e_j))$: $e_j \rightarrow N_i$ (identification) способный определить вероятность/ меру сходства идентифицируемого исполняемого файла e_j с именем N_i эталонной программы $P_{\text{этал.},i}$, который бы удовлетворял следующему критерию: *точность* (метрика оценки результатов идентификации: *Accuracy* или *F-measure*) должна быть *максимальной*, при условии ограничений на:

– потенциальное количество различных программ, установленных на одном компьютере или на всех компьютерах организации;

– количество различных версий для каждой программы, измеряемое от одной до десятков, но в практике не превосходящее сотни;

– наличие существенных изменений в различных версиях одной программы;

– наличие индивидуального характера распределения признака идентифицируемого файла от других программ;

– возможность производить дизассемблирование исполняемого файла;

– не способность в выявлении закладки вредоносного кода в легитимный исполняемый файл;

– не способность классификации выдать корректный результат для программы, класс которой не был определен на этапе формирования модели классификации (нельзя создать класс «файл не похож ни на одну из программ»).

Необходимо провести обучение данного алгоритма по обучающей выборке P и протестировать его на тестовой E. При проведении сравнения, в зависимости от выбранной меры схожести, идентифицируемый исполняемый файл будет считаться программой N_i, при достижении алгоритмом I порогового значения (статистические критерии) или же максимальной вероятности принадлежности к классу (алгоритмы машинного обучения).

При решении задачи идентификации исполняемых файлов на качество идентификации могут оказывать влияние следующие факторы:

– используемая характеристика исполняемого файла;

– метод сравнения СИИФ и ЭСП.

Таким образом, основными подзадачами идентификации исполняемых файлов, можно выделить следующие:

– выделение информативных характеристик ELF-файлов, обладающих максимальной различающей способностью;

– разработка метода идентификации исполняемого ELF-файла по его внутренним характеристикам, обеспечивающего максимальную точность идентификации, при заданных ограничениях.

Из приведенных подзадач можно выделить более частные задачи исследования:

– разработка метода выделения признакового пространства программного обеспечения с последующим созданием сигнатур исполняемых файлов на его основе;

– разработка метода сравнения СИИФ с ЭСП, обеспечивающий более высокую точность идентификации;

– разработка методики идентификации версий программного обеспечения на основе созданного архива сигнатур с использованием статистического подхода и машинного обучения;

– проведение вычислительного эксперимента и обоснование применимости разработанного метода по идентификации программного обеспечения.

Выводы по главе 2

а) Проведенный анализ методов идентификации ПО и выявленные их ограничения и недостатки, позволяют сделать вывод о том, что для задачи данного исследования наиболее подходящим будет разработка подхода к идентификации на основе статического анализа характеристик программ и использование статистических и машинных алгоритмов классификации.

б) Анализ зарубежных и отечественных работ позволяет сделать вывод, что методы машинного обучения дают наиболее высокий показатель точности идентификации ПО.

в) В сфере идентификации ПО большая часть исследований направлена на задачу классификации вредоносного ПО, имеющего дополнительные отличительные черты функциональности, облегчающие задачу его анализа. Однако перед настоящим исследованием стоит более трудная задача по идентификации не вредоносного ПО.

г) На основании предыдущих пунктов и проведенного анализа литературы можно утверждать, что на данный момент не существует решения, позволяющего с максимальной точностью производить идентификацию не вредоносного ПО на основе его характеристик. Произведенный анализ позволил определить основные

задачи в области идентификации ПО и обозначил необходимость проведения исследования, направленного на максимизацию точности идентификации исполняемых файлов, путем совершенствования существующих подходов, поиска новых, ранее не применявшихся, решений в области выделения признакового пространства файлов, методов идентификации, а также способов постобработки результатов.

Глава 3. Разработка методики идентификации установленного программного обеспечения на основе статических характеристик программного кода файла

Разработка методики идентификации исполняемых файлов включает в себя следующие задачи:

– исследование исполняемых файлов в соответствии со стандартом представления формата ELF, проведение их структурного и семантического анализа с целью формирования представления о процессе исполнения программ, их расположения в АС;

– изучение способов представления исполняемых файлов, выделение списка характеристик, пригодных для использования при формировании сигнатур исполняемых файлов, их анализ и выбор наиболее информативных;

– формирование модели представления исполняемых файлов, исходя из результатов, полученных в предыдущих пунктах;

– разработка метода формирования сигнатур исполняемых файлов, предоставляющий возможность идентификации на их основе программ, ранее не задействованных на этапе создания АЭСП. Необходимо сформировать признаковое пространство на основе выбранных характеристик исполняемых файлов, проанализировать существующие подходы к определению их информативности;

– разработка метода сравнения нескольких сигнатур и предоставления идентификатора программы, основываясь на проведенном обзоре зарубежных и отечественных исследований. Также следует обозначить подход к оценке вычислительной сложности созданных методов;

– последней решаемой задачей станет компоновка результатов описанных задач и описание методики идентификации программного обеспечения. Описание этапов проведения идентификации, а также процесс оценки точности идентификации ПО.

3.1 Анализ структуры и характеристик исполняемого файла

Исполняемый файл или программа представляет собой комбинацию компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления [77]. Данные исполняемого файла имеют свой формат (например: EXE, PE, ELF, COFF и т.д.) в зависимости от операционной системы, и состоят из нескольких частей, как правило заголовков, самого тела программы – кода, и остального.

В данной работе исследуются исполняемые ELF-файлы операционной системы Linux. Аббревиатура данного формата расшифровывается как Executable and Linkable Format и в переводе означает формат исполнимых и компонуемых файлов.

ELF-файлы включают в себя program linking (собирание программы) и program execution (запуск программы). Для удобства и эффективности данный формат файла предоставляет параллельное представление содержимого, отражающее различные потребности его активности. На рисунке 5 показана организация ELF-файла, где область слева (Linking View) отображает процесс собирания программы, а область справа (Execution View) – процесс ее исполнения.

ELF заголовок располагается в начале файла и содержит, так называемую, «дорожную карту», которая описывает структурную организацию файла. Также стоит отметить, что первые четыре байта отведены для «магического числа» – идентификатора, имеющего значение 7F 45 4C 46, где 7F – префикс, а остальные расшифровываются в соответствии с ISO/IEC 8859-1 как 45 = E,4C = L,46 = F. *Секции* содержат основную часть информации исполняемого файла для сборки программы: инструкции, данные, таблицы символов, информацию о перемещении и т.д. *Таблица заголовков программы* сообщает системе, как создать образ процесса. Файлы, используемые для создания образа процесса (исполнения программы), должны иметь таблицу заголовков программы; перемещаемые файлы в ней не нуждаются. *Таблица заголовков секций* содержит информацию, описывающую секции файла. Каждая секция имеет запись в таблице, которая

предоставляет такую информацию, как название секции, размер секции и т.д. Файлы, используемые при собирании программы, должны иметь таблицу заголовка раздела; остальные файлы могут не иметь ее.

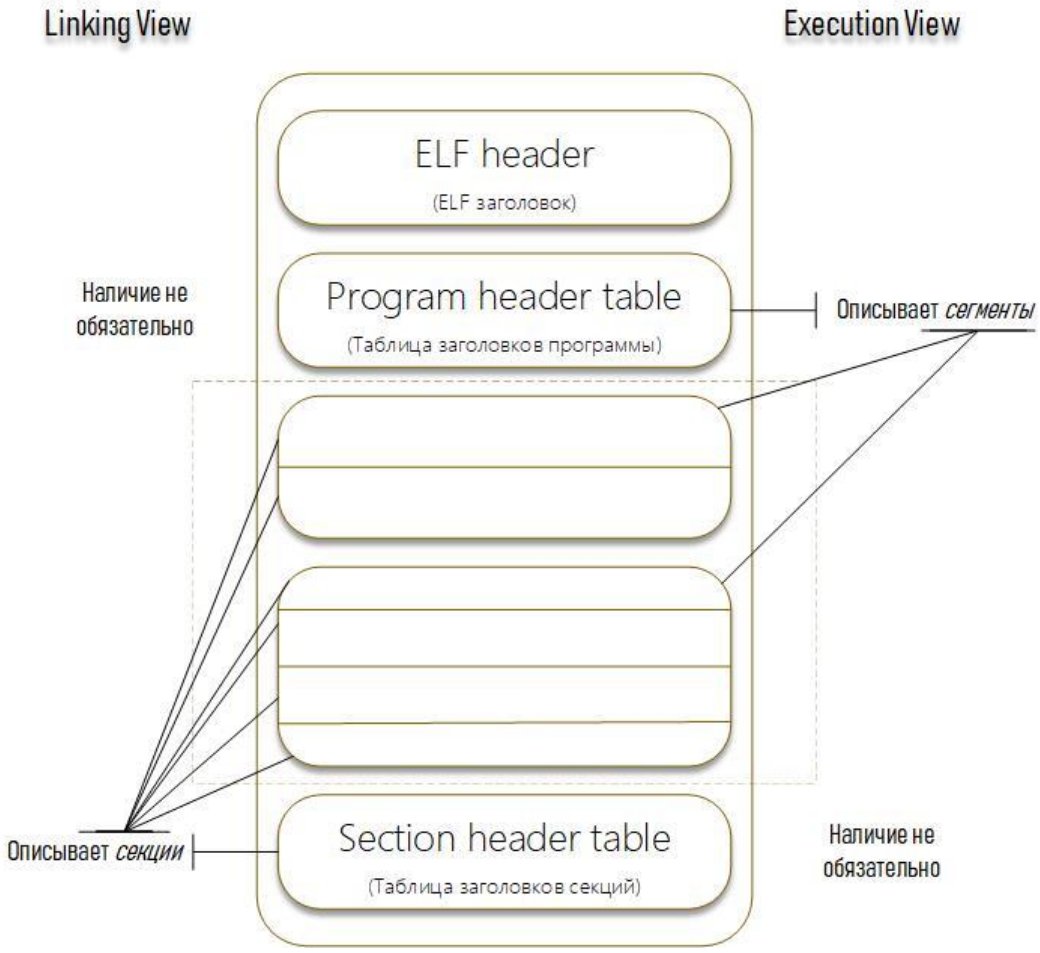


Рисунок 5 – Представление ELF-файла

С помощью команды *readelf* можно посмотреть на структуру и состав ELF-файла. Ввод данной команды в окне консоли выдаст следующую информацию:
Команда для отображения ELF заголовка:

readelf --file-header htop

```
Magic:          7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF32
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           Intel 80386
```

```
Version:                                   0x1
Entry point address:                       0x804ee80
Start of program headers:                  52 (bytes into file)
Start of section headers:                  122368 (bytes into file)
Flags:                                     0x0
Size of this header:                       52 (bytes)
Size of program headers:                   32 (bytes)
Number of program headers:                 9
Size of section headers:                   40 (bytes)
Number of section headers:                 28
Section header string table index:
                                           27
```

Команда для отображения таблицы заголовков программы:

readelf --program-headers htop

```
Elf file type is EXEC (Executable file)
Entry point 0x804ee80
There are 9 program headers, starting at offset 52

Program Headers:
Type   Offset    VirtAddr    PhysAddr    FileSiz  MemSiz   Flg  Align
PHDR   0x000034  0x08048034  0x08048034  0x00120  0x00120  R E  0x4
INTER  0x000154  0x08048154  0x08048154  0x00013  0x00013  R    0x1
P
LOAD   0x000000  0x08048000  0x08048000  0x1c17c  0x1c17c  R E  0x1000
LOAD   0x01cef0  0x08065ef0  0x08065ef0  0x00e08  0x019f0  RW   0x1000
DYNAM  0x01cefc  0x08065efc  0x08065efc  0x00100  0x00100  RW   0x4
IC
NOTE   0x000168  0x08048168  0x08048168  0x00044  0x00044  R    0x4
GNU_E  0x0190dc  0x080610dc  0x080610dc  0x0073c  0x0073c  R    0x4
H_FRA
ME
GNU_S  0x000000  0x00000000  0x00000000  0x00000  0x00000  RW   0x4
TACK
GNU_R  0x01cef0  0x08065ef0  0x08065ef0  0x00110  0x00110  R    0x1
ELRO

Section to Segment mapping:
Segment Sections...
```

```
00
01  .interp
02  .interp .note.ABI-tag.note.gnu.build-id .gnu.hash .dynsym .dynstr
    .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text
    .fini .rodata  .eh_frame_hdr .eh_frame
03  .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss

04  .dynamic
05  .note.ABI-tag .note.gnu.build-id
```

```
06  .eh_frame_hdr
07
08  .init_array .fini_array .jcr .dynamic .got
```

Команда для отображения таблицы заголовков секций:

readelf --section-headers htop

```
There are 28 section headers, starting at offset 0x1de00:

Section Headers:

[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
[ 0]                   NULL            00000000 000000 000000 00      0   0  0
[ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
[ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
[ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
[ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 00076c 04   A  5   0  4
[ 5] .dynsym           DYNSYM          08048918 000918 0014d0 10   A  6   1  4
[ 6] .dynstr           STRTAB          08049de8 001de8 00119a 00   A  0   0  1
[ 7] .gnu.version      VERSYM          0804af82 002f82 00029a 02   A  5   0  2
[ 8] .gnu.version_r    VERNEED         0804b21c 00321c 0000a0 00   A  6   2  4
[ 9] .rel.dyn          REL             0804b2bc 0032bc 000038 08   A  5   0  4
[10] .rel.plt          REL             0804b2f4 0032f4 0003b8 08   A  5  12  4
[11] .init             PROGBITS        0804b6ac 0036ac 000024 00  AX  0   0  4
[12] .plt              PROGBITS        0804b6d0 0036d0 000780 04  AX  0   0  16
[13] .text             PROGBITS        0804be50 003e50 012cb8 00  AX  0   0  16
[14] .fini             PROGBITS        0805eb08 016b08 000015 00  AX  0   0  4
[15] .rodata           PROGBITS        0805eb20 016b20 0025bc 00   A  0   0  32
[16] .eh_frame_hdr     PROGBITS        080610dc 0190dc 00073c 00   A  0   0  4
[17] .eh_frame         PROGBITS        08061818 019818 002964 00   A  0   0  4
[18] .init_array       INIT_ARRAY      08065ef0 01cef0 000004 00  WA  0   0  4
[19] .fini_array       FINI_ARRAY      08065ef4 01cef4 000004 00  WA  0   0  4
[20] .jcr              PROGBITS        08065ef8 01cef8 000004 00  WA  0   0  4
[21] .dynamic          DYNAMIC         08065efc 01cefc 000100 08  WA  6   0  4
[22] .got              PROGBITS        08065ffc 01cffc 000004 04  WA  0   0  4
[23] .got.plt          PROGBITS        08066000 01d000 0001e8 04  WA  0   0  4
[24] .data             PROGBITS        08066200 01d200 000af8 00  WA  0   0  32
[25] .bss              NOBITS          08066d00 01dcf8 000be0 00  WA  0   0  32
[26] .gnu_debuglink    PROGBITS        00000000 01dcf8 00000c 00      0   0  1
[27] .shstrtab         STRTAB          00000000 01dd04 0000fc 00      0   0  1
```

```
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude), p
(processor specific)
```

Команда для отображения содержимого секции в шестнадцатеричном формате представления:

readelf --hex-dump .data htop

```
Hex dump of section '.data':
  0x08066200 00000000 00000000 00000000 00000000 ................
  0x08066210 00000000 00000000 00000000 00000000 ................
  0x08066220 8cef0508 8cef0508 8cef0508 88eb0508 ................
  0x08066230 8cef0508 8cef0508 8feb0508 96eb0508 ................
  0x08066240 9deb0508 a4eb0508 00000000 00000000 ................
  0x08066250 00000000 00000000 00000000 00000000 ................
  0x08066260 8cef0508 8cef0508 8cef0508 8cef0508 ................
  0x08066270 abeb0508 b2eb0508 8cef0508 8cef0508 ................
  0x08066280 8cef0508 a4eb0508 00000000 00000000 ................
  0x08066290 00000000 00000000 00000000 00000000 ................
...
```

3.1.1 Представление исходного кода на языке ассемблера

Структура ELF-файла была подробно рассмотрена в предыдущем пункте, однако любой файл имеет представление и в других форматах, например, в виде бинарной последовательности, декомпилированного или дизассемблированного кода, набора последовательностей выполняемых функций и т.д.

Как было отмечено ранее, в настоящем исследовании в качестве идентификационных характеристик исполняемых файлов были выбраны ассемблерные команды, таким образом, ниже будет приведено представление программы *Htop* в ассемблерном коде.

Язык ассемблера представляет собой низкоуровневый язык машинно-ориентированного программирования, его команды прямо соответствуют отдельным командам машины или их последовательностям. Синтаксис, в общем виде, представляет собой последовательность из адреса – команды – значения.

Представление части программы *Htop* в дизассемблированном виде, где команды выделены синим цветом:

```
htop:     file format elf32-i386
Disassembly of section .interp:
```

```
<.interp>:
2f                      das
6c                      insb   (%dx),%es:(%edi)
69 62 2f 6c 64 2d 6c    imul   $0x6c2d646c,0x2f(%edx),%esp
69 6e 75 78 2e 73 6f    imul   $0x6f732e78,0x75(%esi),%ebp
2e 32 00                xor    %cs:(%eax),%al
```

```
Disassembly of section .data:
<__data_start>:
```

```
...
8c ef                   mov    %gs,%edi
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 88 eb    or     %cl,-0x1477f7fb(%edi,%ebp,8)
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 8f eb    or     %cl,-0x1470f7fb(%edi,%ebp,8)
05 08 96 eb 05          add    $0x5eb9608,%eax
08 9d eb 05 08 a4       or     %bl,-0x5bf7fa15(%ebp)
eb 05                   jmp    806624c
08 00                   or     %al,(%eax)
...
00 00                   add    %al,(%eax)
00 8c ef 05 08 8c ef    add    %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 ab eb    or     %cl,-0x1454f7fb(%edi,%ebp,8)
05 08 b2 eb 05          add    $0x5ebb208,%eax
08 8c ef 05 08 8c ef    or     %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 a4 eb 05 08 00 00    or     %ah,0x805(%ebx,%ebp,8)
...
00 00                   add    %al,(%eax)
8c ef                   mov    %gs,%edi
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 8c ef    or     %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 8c ef    or     %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05          add    $0x5ef8c08,%eax
08 8c ef 05 08 a4 eb    or     %cl,-0x145bf7fb(%edi,%ebp,8)
05 08 00 00 00          add    $0x8,%eax
...
```

```
00 8c ef 05 08 8c ef   add    %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05         add    $0x5ef8c08,%eax
08 8c ef 05 08 8c ef   or     %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05         add    $0x5ef8c08,%eax
08 8c ef 05 08 8c ef   or     %cl,-0x1073f7fb(%edi,%ebp,8)
05 08 8c ef 05         add    $0x5ef8c08,%eax
08 a4 eb 05 08 00 00   or     %ah,0x805(%ebx,%ebp,8)
...
```

3.1.2 Анализ особенностей ассемблерного кода программного обеспечения

Представление программы в том или ином виде (бинарном, ассемблерном, на уровне инструкций высокоуровневого языка программирования, структуры формата файла и других), обладает некоторыми особенностями, которые будут рассмотрены ниже.

Уровень абстракции. Наличие смысловых конструкций, упрощающих понимание структуры данных и операций над ними для человека. Высокий уровень абстракции используется в высокоуровневых языках программирования для описания структур данных и операций над ними, описание которых на машинном коде очень длинны и сложны для понимания человеком. Здесь присутствует широкий выбор команд, а имена переменных задаются программистом, в дальнейшем такой программный код компилируется и переводится на низкоуровневый язык (например, ассемблер или байт-код). Чем выше уровень абстракции, тем разнообразнее можно реализовать некоторую функцию в программе, необходимо принимать больше усилий в детальном исследовании всех процедур и данных программы. В то же время, представление исполняемого файла на низкоуровневом языке программирования значительно сокращает число возможных используемых значений признака, в том числе таких, чьи функциональные назначения носят схожий характер. Такой код становится легче анализировать и автоматизировать подход к идентификации, не привлекая к работе человека;

Диапазон признака. Количество вариаций некоторой характеристики файла носит ограниченный характер. Известно, что степень свободы в статистике – число

количества значений, которые могут варьироваться, играет важную роль в количестве информации, носимой признаковым пространством. Так при рассмотрении признака, имеющего три градации (степень свободы здесь равна двум) количество информации, которое будет носить построенная на нем сигнатура, будет ниже, чем при использовании признака с десятью градациями (степень свободы здесь равна девяти);

Уникальность и информативность значения признака. Значение признака может не носить никакой информативной составляющей, как например, значение байта «00», присутствие которого во всех файлах носит скорее заполняющий характер, нежели информационный. Очевидно, что некоторые значения признака будут обладать более существенными идентифицирующими способностями, таковыми обычно являются редко встречающиеся градации признака, они и обладают наибольшей информативностью. Однако в рамках разрабатываемой методики идентификации следует учитывать присущую разнообразность исполняемым файлам, участвующим в процессе аудита электронных носителей информации. Программы отличаются не только функциональной направленностью, но и объемом, что существенно влияет на вид формируемых сигнатур при использовании разработанного метода, описанного в разделе 2.3. Если в качестве признака выбрать редко встречаемую характеристику файла, то формируемая сигнатура будет содержать большое число нулей, не несущих никакой важной информации. При этом есть большая вероятность, что для исполняемых файлов небольшого объема и сильно отличающихся функциональностью, выбранный признак и вовсе будет давать нулевую сигнатуру, анализ и сравнение которой будет невозможно.

В данной работе производится исследование представления исполняемых файлов в их ассемблерном виде. Так же коды программ рассматриваются как целый объем, так и разделенный на определенные части.

Анализ программы как целого объема данных имеет некоторое преимущество, выраженное в том, что формируемая сигнатура независима от внутреннего расположения блоков данных программы, это актуально для случая

выпуска новой версии программы, содержащей одну или несколько частей измененного кода. При разбиении исследуемого кода файла на части и исследование признака на каждой из них, уже не будет обладать данным преимуществом. Однако анализ распределения признака по коду программы является достаточно информативным показателем.

3.2 Подход к представлению программного обеспечения

В исследование приняло участие 578 исполняемых файла ОС Linux, из них обучающей выборке принадлежало 455 файла различных версий и разрядностей (32х и 64х), относящихся к 63 различным программам. К тестовой выборке же принадлежало 123 файла, относящихся к тем же 63 программам, все они были отличны от задействованных файлов, используемых в обучающей выборке, и имели 32х и 64х разрядности. Весь объем файлов был сформирован формировалась путем скачивания программ с официальных репозиториев ОС Linux для архитектур процессоров x86 и x86-64. Затем, для формирования тестовой выборки она отделялась от обучающей путем извлечения по два исполняемых файла различных версий и разрядностей по каждой программе (кроме одной, представленной только в одной разрядности) из всего объема скаченых файлов.

Стоит отметить, что архив сигнатур не является фиксированным и запрещенным для добавления новых сигнатур, наоборот, в реальных условиях при проведении аудита электронных носителей информации, он должен периодически обновляться и иметь актуальные данные для выполнения поставленных исследователем задач.

Важным также является количество программ и исполняемых файлов, участвующих в процессе аудита. Очевидно, что компьютер пользователя или же совокупность всех компьютеров организации, не смотря на все разнообразие существующих, содержит ограниченное число программ. На основе легитимных из них происходит формирование АЭСП. Также учитывая тот факт, что каждая из программ реализовывается в различных версиях, число которых часто не

превышает и двух десятков, работа в данной области неизбежно происходит на органиченном наборе данных, увеличние которого в практических реалиях просто не представляется возможным.

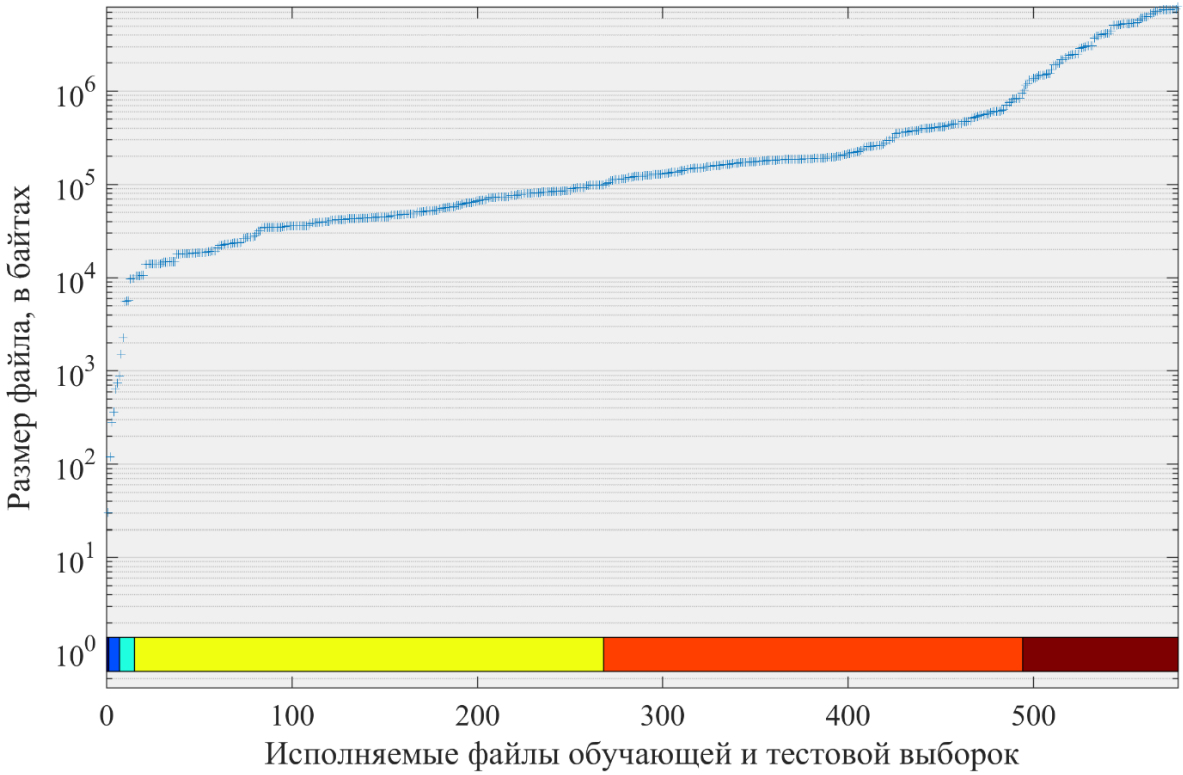


Рисунок 6 – Распределение исполняемых файлов по их размеру

Выше на рисунке 6 представлено распределение размеров участвующих в исследовании программ. Как видно их объем измеряется от десятков байтов, до тысяч. Каждый маркер на рисунке отображает один файл из 578 скаченных файлов. Данные приведены в логарифмической шкале и видно, что наиболее большое число исполняемых файлов имеет размер в пределах 10^4 – 10^6 байтов (желтая и оранжевая области линейчатой диаграммы с накоплением), так средним показателем является $7{,}24 \cdot 10^5$ или 707 Кб.

Необходимо подчеркнуть, что весь объем исследуемых файлов подчиняется закону Бенфорда, который описывает вероятность появления определенной первой значащей цифры в распределениях величин, взятых из реальной жизни [78,79]. Считается, что компьютерные файлы подчиняются этому закону [80], таким образом можно утверждать о разнообразии и репрезентативности изучаемого

объема файлов. В таблице представлены два распределения - Бенфорда и первых знаяащих цифр исследуемых файлов. Так, на основании критерия однородности хи-квадрат и уровне значимости $p = 0{,}05$, можно утверждать, что расхождения между теоретическим и экспериментальным распределениями вероятностей первой цифры для десятичной системы счисления, статистически недостоверны.

Таблица 3 – Сравнение двух распределений

Первая значащая цифра	1	2	3	4	5	6	7	8	9
Распределение Бенфорда	0,301	0,176	0,125	0,097	0,079	0,067	0,058	0,051	0,046
Распределение первых значащих цифр исследуемых файлов	0,336	0,111	0,123	0,130	0,093	0,050	0,073	0,048	0,036

3.2.1 Модель представления исполняемого файла

Исходя из понятий векторной модели, каждый исполняемый файл можно представить в качестве точки n-мерного пространства действительных чисел, т.е. вектора, где n – количество градаций признака и являющееся одинаковым для всех рассматриваемых исполняемых файлов.

Если ввести в рассмотрение n-мерное признаковое пространство $F = (f_1, f_2, \ldots, f_n)$, где $i = 1, 2, \ldots, n$ – число градаций признака, тогда каждый объект o (файл v_m или e_j) в данном пространстве будет отображаться точкой с координатами $o = (f_1(o), f_2(o), \ldots, f_n(o))$, а каждый класс объектов, т.е. имя программы $P_{\text{этал.},i}$ множеством таких точек.

Любой исполняемый файл может быть представлен как совокупность некоторых количественных значений. Ранее было выделено, что наиболее подходящим способом сбора характеристик ПО в данном исследовании является статический сбор частотных характеристик. Нескольким версиям одной и той же программы свойственны определенные характерные особенности. Измерение частот этих особенностей – градаций признака – позволяет нам представить исполняемый файл, как последовательность частот характеристик и перенести его вектор в пространство. Исполняемому файлу может быть поставлена в

соответствие точка в пространстве, координатами которой являются частоты его градаций признака.

Считая, что идентифицируемый файл обладает некоторыми свойственными ему характеристиками, можно сопоставить ему определенные характеристики конкретных программ. Идентификация файла происходит путем применения выбранного алгоритма сравнения к двум векторам значений признака объектов: *o* – идентифицируемого и *o'* – эталонного.

3.2.2 Идентификационное признаковое пространство

В качестве значений идентифицирующих признаков выступают статистические характеристики кода программ, такие как частота признака на всем объеме кода программы, частота признака в частях кода программы; также сюда входит такая градация признака, как наименование ассемблерной команды (*aaa, aad,…, xor*).

Для проведения аудита электронных носителей информации главной задачей является определение такого набора характеристик ELF-фалов, выступающих в качестве соответствующих ему признаков и впоследствии включаемых в ЭСП. Очевидно, что выявление признаков, позволяющих достичь максимальной точности идентификации, является крайне важным. На качество идентификации могут оказать влияние множество факторов, таких как: размер программы, общее количество проверяемых исполняемых файлов, количество сформированных ЭСП в АЭСП, выбранный признак и его градация, способ построения СП и метод сравнения сигнатур.

В качестве идентификатора сопоставляемого, определенной программе, выступает совокупность выявленных характерных особенностей исполняемых файлов, являющихся версиями данной программы. Важным является тот факт, что последующая версия программы должна носить дополняющий предыдущую версию характер, т.е. если создатель программы решил полностью сменить функциональность программы или внес в нее некоторое критически большое число

изменений, такая программа будет считаться новой и для нее необходимо построение отдельной сигнатуры.

Как было представлено выше исполняемый файл является сложной системой, в основе которой лежат последовательности структур данных и операций над ними. Многие из этих функций проявляются в виде конкретных наборов ассемблерных команд, которые могут быть использованы при решении задачи идентификации ПО, однако, в то же время некоторые характеристики являются однотипными, использующиеся во многих различным между собой программах.

Учитывая возможное количество ПО установленного на компьютере пользователя и возможный список легитимного ПО сохраняющегося в АЭСП в работе предлагается использовать сигнатуры, размер которых согласуется с объемами анализируемых файлов и возможными частотами встречаемости характеристик файлов, а также имеет фиксированную длину для всех сравниваемых исполняемых файлов.

С целью повышения точности идентификации и повышения скорости проведения сравнения сигнатур, в работе предложено и обосновано применение новых методов по формированию СИИФ и ЭСП из признаков, а также построение унифицированных сигнатур (УС), рассмотрение которых будет в пункте 2.3.2.

Возможными характеристиками эталонных программ являются наименования ассемблерных команд *As.com*. Так, в качестве совокупности характеристик версии программы могут выступать:

– $v_m = (As.com)$, где $As.com = (ac_1, ac_2, \ldots, ac_{118})$ – последовательность из 118 отобранных ассемблерных команд. Сигнатура представляет собой частотную последовательность 118 ассемблерных команд на всем объеме кода программы;

– $v_m = (ac_{n,1}, ac_{n,2}, \ldots, ac_{n,k})$, где $ac_{n,k}$ – n-ая выбранная ассемблерная команда из 118 ассемблерных команд $ac_n \in As.com$, k – число интервалов, на которые был разделен код программы. Сигнатура представляет собой частотную

последовательность выбранной ассемблерной команды в определенных интервалах кода программы.

3.2.3 Модель представления программного обеспечения

Идентифицируемый исполняемый файл обладает характеристиками, свойственными некоторой программе. Таким образом, можно детерминировать некоторую программу $P_{этал.,i}$ как совокупность ее выпущенных версий v_m, представляемых в виде набора, координатам которого сопоставляется множество характеристик F. Схематическое отображение модели представления программного обеспечения изображено на рисунке 7.

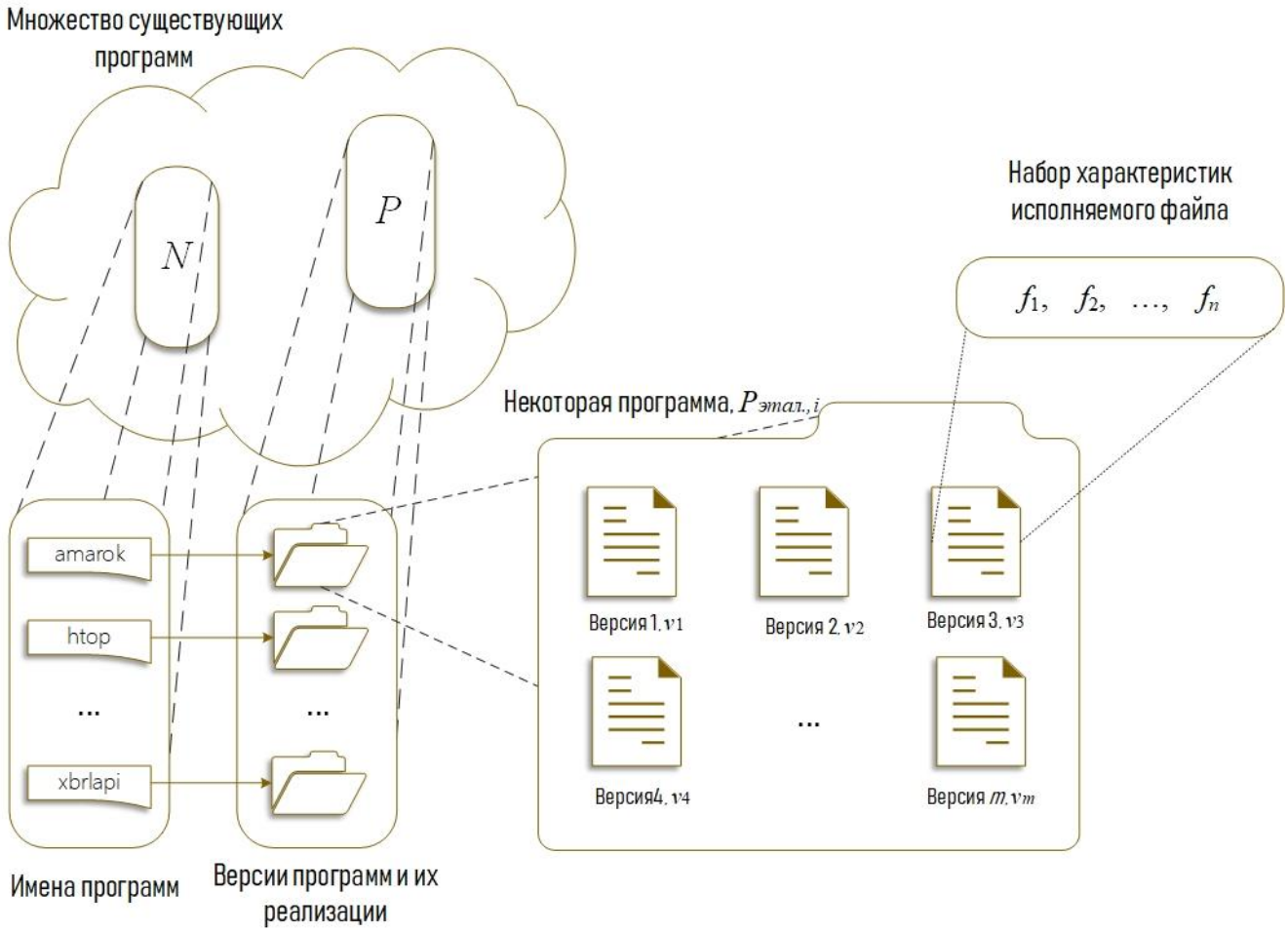


Рисунок 7 – Модель представления программного обеспечения

3.3 Метод формирования сигнатур исполняемых файлов, обладающий достаточной гибкостью по распознаванию различных версий программ, а также позволяющий наиболее точно производить идентификацию

Предлагается производить отбор признаков, которые обладают наиболее высокой различающей способностью и обеспечивающие достаточную частоту встречаемости для формирования не нулевых сигнатур исполняемых файлов.

В исследовании рассматривается несколько подходов к идентификации ELF-файлов, каждый из которых обладает своими ограничениями, которые следует учитывать еще на этапе формирования сигнатур файлов. Так, в данном разделе будут описаны следующие подходы к формированию сигнатур:

– Формирование унифицированных сигнатур программ, объединяющих в себе несколько частотных характеристик различных файлов версий одной программы. Решение о группировке тех или иных частотных последовательностей принимается на основе метода кластеризации *k*-средних. Сигнатуры самих исполняемых файлов строятся на одном из следующих подходов:

1) подсчет распределения частоты появления 118 ассемблерных команд на всем объеме дизассемблированного кода файла;

2) подсчет распределения частоты появления одной ассемблерной команды на *k* не равных интервалах дизассемблированного кода файла;

3) подсчет распределения частоты появления одной ассемблерной команды на *k* равных интервалах дизассемблированного кода файла.

– Формирование сигнатур по каждым версиям программ, которые строятся на подсчете распределения частоты появления одной ассемблерной команды на *k* равных интервалах дизассемблированного кода файла.

Среди всего многообразия ассемблерных команд были отобраны 118, которые присущи файлам обучающей выборки. Однако, в ходе работы над исследованием, было решено проанализировать информативность этих команд – данный этап описан в пункте 2.3.1, и произвести отбор идентификационных признаков с целью разработки метода по формированию СП на основе

распределения ассемблерной команды в определенных интервалах кода программы.

На рисунке 8 представлены графики распределений частот выбранных ассемблерных команд для одной версии программы *Amarok*. На оси абсцисс отмечены номера интервалов, на которые был разделен код программы k от 1 до 30, а на оси ординат – частоты встречаемости той или иной ассемблерной команды ac_i.

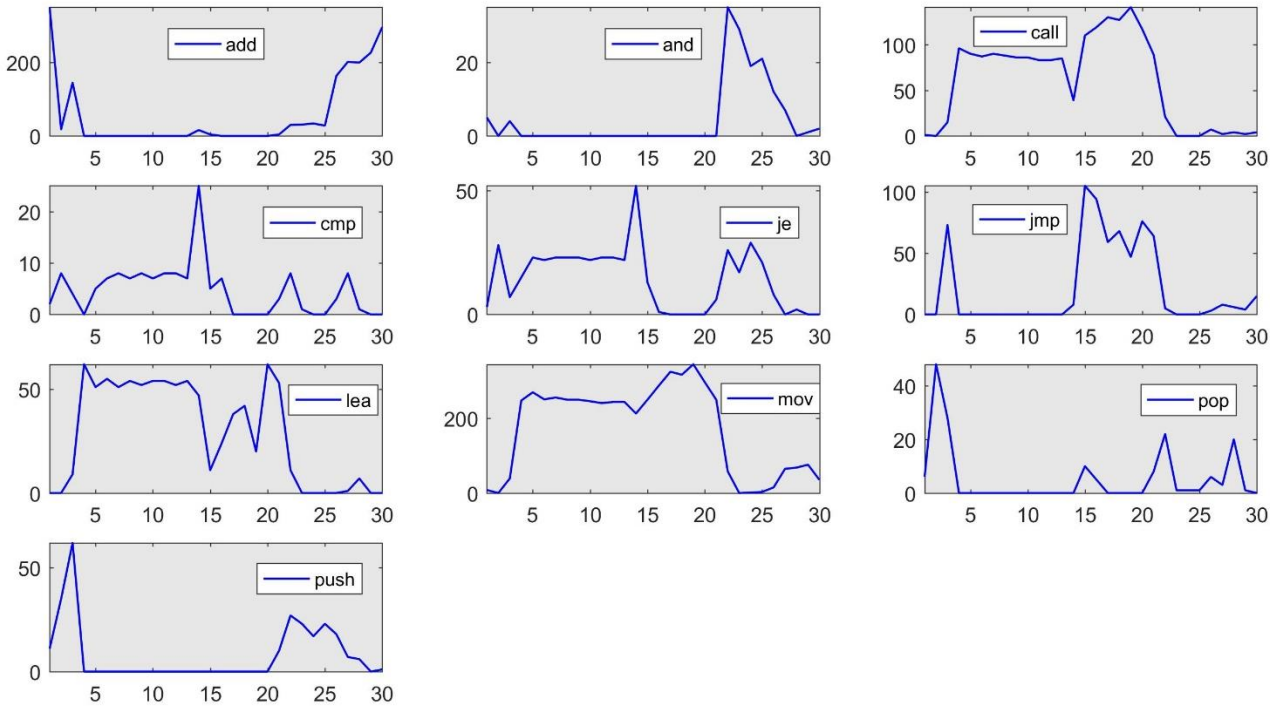


Рисунок 8 – Частотные распределения ассемблерных команд.

Из рисунка 8 видно, что характеры распределений разных ассемблерных команд различаются между собой, одни команды располагаются в основном в середине программы, другие в начале и в конце.

3.3.1 Критерии и алгоритм отбора наиболее информативных признаков

Информативность признака может выражаться в различных величинах, в зависимости от используемого подхода:

– энергетический – где информативность оценивается по величине признака. Здесь наиболее информативным признаком будет считаться тот, чье абсолютное значение наибольшее. Из этого становится понятным, что использование энергетического подхода к задаче идентификации ПО не пригодно, ведь в ситуации, когда значение какого-либо признака велико по абсолютной величине и практически не отличается у файлов, относящихся к различным программам, идентификация объекта к какому-либо одному классу является практически невозможной;

– информационный – где информативность признака оценивается по величине достоверного различия между классами образов в пространстве признаков.

Наиболее пригодным подходом для оценивания информативности признаков, является информационный подход, позволяющий производить отбор признаков, обладающих максимальной различающей способностью именно на определенной группе программ, задействованных в формировании АЭСП.

Одним из ключевых вопросов является выбор алгоритма вычисления информативности 118 ассемблерных команд. Основываясь на проведенных исследованиях [81], [82], был выбран метод Шеннона, не зависящий от объема выборок наблюдаемых признаков, а также он позволяет вычислить информативность признака для произвольного числа классов. Данный алгоритм оценивает информативность как средневзвешенное количество информации (величину устраненной энтропии), свойственное рассматриваемой градации признака $f \in F$ и предоставляет оценку информативности $I(f)$ в виде нормированной величины, принимающей значения в интервале от нуля (отсутствие информативности) до единицы (высокая степень информативности).

Для оценки информативности признака f пользуются следующей формулой:

$$I(f) = 1 + \sum_{i=1}^{n} \left(P_i \cdot \sum_{u=1}^{U} P_{i,u} \cdot \log_U P_{i,u} \right), \quad (1)$$

где n – число градаций признака; U – число программ, участвующих в подсчете информативности; P_i – вероятность попадания значения признака в i-ю градацию, которая вычисляется по формуле:

$$P_i = \frac{\sum_{u=1}^{U} m_{i,u}}{N},$$

где $m_{i,u}$ – частота появления значения признака в i-й градации в u-ом классе; N – общее число наблюдений признака; $P_{i,u}$ – вероятность появления i-й градации признака в k-ой программе, которая вычисляется по формуле:

$$P_{i,u} = \frac{m_{i,u}}{\sum_{u=1}^{U} m_{i,u}}.$$

Очевидно, что произвести оценку информативности признаков на всем пространстве существующих программ и их версий, является невозможным, поэтому в расчетах принимало ограниченное число исследуемых файлов. Было отобрано 10 различных программ и версий, из которых было сформировано 52 частотных распределения характеристики – ассемблерной команды, путем подсчета каждой из 118 ассемблерных команд на всем объеме дизассемблированных кодов исполняемых файлов.

Решение задачи по отбору наиболее информативных признаков разбивается на два случая:

– первый, когда мы принимает число градаций признака n равным 1, т.е. появление выбранной ассемблерной команды;

– второй, когда мы принимает число градаций признака n равным 2, т.е. появление выбранной ассемблерной команды и появление другой, отличной от данной, команды.

Далее, в таблицах 4 и 5 приведены результаты расчета информативности для $n = 1$ и $n = 2$ соответственно. Все ассемблерные команды расположены в порядке возрастания их информативности. Полный перечень 118 ассемблерных команд и их информативности по Шеннону можно найти в приложении А.

Таблица 4 – Значения информативности для 118 ассемблерных команд, по методу Шеннона и заданным числом градаций признака $n = 1$

Ассемблерные команды	**Информативность, $I(x)$**
cmpsb, cmpsw, esc, jc, jcxz, jna, jnae, jnb, jnbe, jnc, jng, jnge, jnl, jnle, jnz, jpe, jpo, jz, lodsb, lodsw, loopnz, loopz, movsb, movsw, repe, repne, retn, sal, scasb, scasw, stosb, stosw, wait	0
rep, jle, dec, *lea*, std, *cmp*, shr, *jmp*, jg, shl, *and*, cld, rol, js, jl, *add*, nop, mul	0,1408 – 0,1990
jne, **push**, ret, sub, xor, jge, ror, not, test, *mov*, div, jbe, jb, loopne, clc, *je*, jae, xchg, ja, or, repz, cli, adc, pushf, hlt, sar, sti, sbb, neg, jns, idiv, in, *call*, imul, jp, loop, jno, *pop*, repnz, out, iret, loope, jnp, xlat, int, rcl, cmc	0,2001 – 0,2972
jo, stc, lahf, retf, rcr, sahf, inc, cwd, popf, cbw	0,3026 – 0,3778
lock	0,4228
daa, les, into, aaa, lds, aam, aas, das, aad	0,6144 – 0,6785

Таблица 5 – Значения информативности для 118 ассемблерных команд, по методу Шеннона и заданным числом градаций признака $n = 2$

Ассемблерные команды	**Информативность, $I(x)$**
cmpsb, cmpsw, esc, jc, jcxz, jna, jnae, jnb, jnbe, jnc, jng, jnge, jnl, jnle, jnz, jpe, jpo, jz, lodsb, lodsw, aaa, les, daa, aas, aam, ja, mul, js, loop, jge, jp, hlt, jl, jns, loopne, iret, das, rep, jae, idiv, loopnz, loopz, movsb, movsw, repe, repne, retn, sal, scasb, scasw, stosb, stosw, wait, cwd, cbw, div, neg, into, lds, aad	0,2024
rcl, std, sbb, lahf, ror, ret, jne, cmc, popf, rcr, adc, jno, in, shl, jbe, jb, pushf, not, clc, rol, int, jle, jg, sub, loope, xlat, cld, sti, shr, jnp, repnz, cli, dec, repz	0,2025
imul, sar, sahf, *jmp*, xor, nop, *and*, jo, *je*, stc, test, *push*, retf	0,2026
out, *cmp*, inc	0,2027
or	0,2028
xchg	0,2029
add, *pop*	0,203
lea	0,2032
call	0,2036
mov	0,2044
lock	0,2095

Из приведенных результатов можно сделать вывод, что значение числа градаций несет важную роль, интерпретировать которую следует как результат существенного влияния на параметры P_i и $P_{i,u}$ в формуле (1) дополнительной градации, т.е. не появление рассматриваемого признака, а появление другой, отличной ассемблерной команды. Таким образом, происходит учет отношения числа встречаемости рассматриваемой ассемблерной команды к числу появления

всех остальных 117 ассемблерных команд, что позволяет оценить информативность не только на основе различия частот встречаемости команды в различных классах, но и на основе ее доли по отношению к остальным командам.

Стоит отметить, что использование наиболее информативных команд из таблицы 4 невозможно, т.к. их частота встречаемости в дизассемблированных кодах программ слишком мала для формирования сигнатур с достаточным числом ненулевых значений. Использование второго же подхода с числом градаций признака $n = 2$ позволяет устранить данный недостаток.

Имена ассемблерных команд, выделенные курсивом, были отобраны как наиболее подходящие для решения задачи данной научной работы. Основываясь на результатах таблиц 4 и 5, примем следующий порядок информативности выбранных 10 ассемблерных команд: *mov*, *call*, *pop*, *push*, *je*, *lea*, *add*, *cmp*, *and*, *jmp*.

3.3.2 Формирование унифицированных сигнатур программ (УСП), используемого в дальнейшем в статистическом методе сравнения сигнатур и с использованием искусственной нейронной сети

Формирование УСП, объединяющих в себе несколько частотных характеристик различных файлов версий одной программы. Решение о группировке тех или иных частотных последовательностей принимается на основе метода кластеризации *k*-средних. Среди всех версий одной программы выделяются кластеры, на основе которых происходит формирование одного варианта унифицированной ЭСП.

Подход на основе 118 ассемблерных команд.

Первым подходом к формированию сигнатур исполняемых файлов рассмотрим использование в качестве характеристики F файла ассемблерный код *As.com*, а именно его команды ac_1, ac_2,…, ac_{118} в качестве градаций признака. Таким образом, здесь $F = (f_1, f_2, \ldots, f_n) = As.com = (ac_1, ac_2, \ldots, ac_n)$, $n = 118$ – является признаковым пространством, где значение признака $L(ac_n) \in \mathbb{N}_0$ и показывает количественное значение для градации признака ac_n.

На основе отобранных 118 градаций признака происходит формирование как сигнатур программ обучающей выборки, так и сигнатур идентифицируемых файлов тестовой выборки. Однако, ЭСП будут создаваться путем унификации сигнатур исполняемых файлов обучающей выборки и добавления наименования известной программы.

Пусть обучающей выборке программ $P = \{P_{этал.,1}, P_{этал.,2}, \ldots, P_{этал.,i}\}$ присвоены имена этих программ $N = \{N_1, N_2, \ldots, N_i\}$. Каждая отдельная программа $P_{этал.,i}$ состоит из ряда различных версий $v_1, v_2, \ldots, v_m$, тогда версия v_m будет описываться n градациями признака $F = As.com = (ac_1, ac_2, \ldots, ac_{118})$.

Процесс создания сигнатур всех программ обучающей выборки будет состоять в следующих шагах (описание дается для каждой рассматриваемой программы в отдельности):

1) Каждая версия программы v_m дизассемблируется.

2) Из полученного представления кода выделяется частотная характеристика выбранного признака F (118 ассемблерных команд), являющаяся основой для формирования сигнатуры одного исполняемого файла $S'(v_m) = q(v_m, As.com) = (ac_1, ac_2, \ldots, ac_{118})$, где $q()$ – функция $q(v_m, F)$: $As.com \rightarrow \mathrm{N}_0$, определяющая алгоритм формирования количественного значения для ac_n ассемблерной команды, где $As.com$ – множество принимаемых значений характеристики F. Сигнатура версии программы принимает вид:

$$S'(v_m) = (L(ac_1), L(ac_2), \ldots, L(ac_n)), n = 118,$$

где $L(ac_n)$ – значение частоты встречаемости градации признака ac_n в дизассемблированной коде файла v_m.

Однако, многообразие версий ПО, растущие функциональные возможности компьютерных систем, различные временные промежутки выпуска новой версии ПО и т.п. эти факторы вносят существенные коррективы в содержательную часть кода программ. Так, выпуск новых версий некоторой программы может содержать в себе существенные изменения в отличии от выпущенных ранее версий. Следует учитывать данный нюанс и при последующем формировании унифицированных

сигнатур из сигнатур версий $S'(v_m)$ производить оценку отклонения значения частоты ассемблерной команды $L(ac_n)$ файла v_m от значения средней частоты ассемблерной команды $\bar{L}(ac_n)$ программы $P_{этал,i}$ по формулам:

а) Расчет средних частот $\bar{L}(ac_n)$ для каждой из ассемблерных команд по всем $S'(v_m)$

$$\bar{L}(ac_n) = \frac{1}{m}\sum_1^m S'(v_m)[ac_n],$$

получаем строку со 118-ю значениями: $\bar{S}'(P_{этал,i}) = (\bar{L}(ac_1), \bar{L}(ac_2),\ldots, \bar{L}(ac_n))$.

б) Расчет промежуточной унифицированной сигнатуры $S'_{униф.}(P_{этал,i}) = (p(ac_1), p(ac_2),\ldots, p(ac_n))$

$$p(ac_n) = \begin{cases} \bar{L}(ac_n), \text{если число не нулевых элементов } Q(v_m)[q_n] \geq 0{,}85 \cdot m \\ 0, \text{если число не нулевыхэлементов } Q(v_m)[q_n] < 0{,}85 \cdot m \end{cases}, \quad (2)$$

где $Q(v_m) = (q_1, q_2,\ldots, q_n)$ множество элементов

$$q_n = \begin{cases} L(ac_n), |\bar{L}(ac_n) - L(ac_n)| \leq 0{,}2 \cdot \bar{L}(ac_n) \\ 0, |\bar{L}(ac_n) - L(ac_n)| > 0{,}2 \cdot \bar{L}(ac_n) \end{cases},$$

таких же, как и в сигнатуре версии программы $S'(v_m)$, но обнуляемые при невыполнении условия о попадании частоты ассемблерной команды $L(ac_n)$ в заданный интервал, ограниченный величиной допустимого отклонения от среднего показателя частоты ассемблерной команды для программы $\bar{L}(ac_n)$.

Смысл формулы (2) состоит в следующем: если количество ненулевых значений q_n будет не менее 85% от общего числа файлов v_m в выборке программы $P_{этал,i}$, то в промежуточную унифицируемую сигнатуру $S'_{униф.}(P_{этал,i})$ записывается средняя частота встречаемости каждой из ассемблерных команд для всех версий $S'(v_m)$.

Коэффициенты 0,2 и 0,85 в формулах были вычислены экспериментальным путем.

3) Окончательно унифицированная сигнатура программы имеет вид:

$$S_{униф.}(P_{этал,i}) = (N_i, S'_{униф.}(P_{этал,i})) = (N_i, p(ac_1), p(ac_2),\ldots, p(ac_n)).$$

Все сформированные таким образом унифицированные сигнатуры эталонных программ помещаются в АЭСП, для дальнейшего обращения к нему либо в процессе идентификации, либо при необходимости модификации сигнатур.

В соответствии с описанными выше этапом под пунктом 2) сигнатура идентифицируемого файла имеет вид:

$$S(e_j) = (L(ac_1), L(ac_2),\ldots, L(ac_n)),\ n = 118.$$

Подход на основе одной ассемблерной команды на неравных интервалах.

В разделе 2.3 было приведено обоснование цели исследования подхода к формированию сигнатур программ на основе одной ассемблерной команды. Здесь и в следующем подпункте мы рассмотрим данный подход.

В отличие от способа построения УСП по частотам встречаемости в дизассемблированном коде 118 различных ассемблерных команд, следующий подход к исследованию характеристик исполняемых файлов основан на анализе одной ассемблерной команды, выбранной в качестве признака для формирования частотных распределений. Таким образом, здесь F представляет собой распределение ассемблерной команды на интервалах ассемблерного кода и $F = (f_1, f_2,\ldots, f_n) = ac_n = (ac_{n,1}, ac_{n,2},\ldots, ac_{n,k})$, ac_n – ассемблерная команда $ac_n \in As.com$, k – число интервалов разбиения дизассемблированного кода программы $P_{этал,i}$, где значение признака $L(ac_n) \in \mathbb{N}_0$ и показывает количественное значение для признака ac_n.

На основе отобранного признака происходит формирование как сигнатур программ обучающей выборки, так и сигнатур идентифицируемых файлов тестовой выборки. Однако, ЭСП будут создаваться путем унификации сигнатур исполняемых файлов обучающей выборки и добавления наименования известной программы.

Пусть обучающей выборке программ $P = \{P_{этал.,1}, P_{этал.,2}, \ldots, P_{этал.,i}\}$ присвоены имена этих программ $N = \{N_1, N_2,\ldots, N_i\}$. Каждая отдельная программа $P_{этал.,i}$

состоит из ряда различных версий v_1, v_2,…, v_m, тогда версия v_m будет описываться одной градацией признака *As.com* на n интервалах разбиения дизассемблированного кода программы $F = ac_n = (ac_{n,1}, ac_{n,2},\ldots, ac_{n,k})$.

Опишем процесс формирования сигнатур файлов обучающей выборки:

1) Первоначально фиксируется один из файлов v_{m0} версии программы и происходит его дизассемблирование.

2) Полученный ассемблерный код программы разбивается на интервалы неравных длин, но с фиксированной частотой появления в них заданного признака ac_n (ассемблерной команды). В этом случае за длину интервала принимается количество различных ассемблерных команд из *As.com* на промежутке фиксированной частоты признака $L(ac_n)$. Таким образом, формируется частотная характеристика фиксированного фала v_{m0}, имеющая вид

$$S'(v_{m0}) = ((ac_n, h_1), (ac_n, h_2),\ldots, (ac_n, h_k)),$$

где ac_n – заданная частота признака, равная константе; h_i – длина интервала разбиения; k – количество полученных интервалов разбиения.

Формирование $S'(v_{m0})$ необходимо для того, чтобы стало возможным построение сигнатуры отдельной программы v_m с фиксированным количеством интервалов разбиения k.

3) К остальным файлам версий ПО из обучающей выборки применяется следующая формула:

$$S'(v_m) = ((ac_{n,1}, h'_1), (ac_{n,2}, h'_2),\ldots, (ac_{n,k}, h'_k)),$$

где $ac_{n,k}$ – получаемая частота признака; $h'_k = \frac{h_k \cdot l'_m}{l}$ – длина интервала разбиения, l – длина v_{m0}-го файла, h_k – длина k-го интервала разбиения v_{m0}-го файла, l'_m – длина m-го файла v_m; k – количество полученных интервалов разбиения.

Таким образом, получаются распределения фиксированного файла $S'(v_{m0})$ и остальных файлов $S'(v_m)$ обучающей выборки, с одинаковой длинной k по отдельной программе $P_{\text{этал.},i}$, из которых затем, на основе средних значений частот признака

$$\bar{L}(ac_n) = \frac{1}{m}\left(S'(v_{m0})[ac_n] + \sum_1^m S'(v_m)[ac_n]\right)$$

4) формируется УСП:

$$S_{униф}(P_{этал,i}) = (N_i, \text{S'}(P_{\text{этал},i})) =$$
$$(N_i, \bar{L}(ac_1), h_1), (\bar{L}(ac_2), h_2),\ldots, (\bar{L}(ac_k), h_k)).$$

Все сформированные таким образом сигнатуры помещают в АЭСП.

Особенностью сигнатур идентифицируемых файлов является то, что их длины должны совпадать с длиной сигнатуры соответствующей программы из архива, поэтому сигнатура идентифицируемого файла представляет собой следующую структуру в соответствии с описанными выше этапом под пунктом 3):

$$S(e_j) = ((ac_{n,1}, h'_1), (ac_{n,2}, h'_2),\ldots, (ac_{n,k}, h'_k))$$

и является распределением частот ассемблерной команды в интервалах, заданных в сигнатуре программы из архива, где ac_n – частота признака; $h'_k = \frac{h_k \cdot l'}{l}$ – длина интервала разбиения, l – сумма длин интервалов h_k, заданных в сигнатуре соответствующей программы из архива $S_{униф}(P_{этал,i})$, h_k – длина k-го интервала разбиения сигнатуры программы из архива $S_{униф}(P_{этал,i})$, l' – длина идентифицируемого файла e_j.

Заметим, что предложенный особый метод формирования сигнатур программ имеет некоторые нюансы:

– во-первых, каждая сигнатура в архиве является статистической оценкой (аналогом) равномерного распределения частот некоторой ассемблерной команды, которая осуществлена на основе выборки, отличной от выборки частот идентифицируемого файла;

– во-вторых, для каждого идентифицируемого файла преобразуется его длина относительно сравниваемой с ним программы, что приводит к одинаковой размерности сравниваемых сигнатур;

– в-третьих, для каждой ЭСП из архива необходимо стоить свою СИИФ.

Подход на основе одной ассемблерной команды на равных интервалах.

Процедура формирования сигнатур исполняемых файлов схожа с выше описанной, с тем лишь отличием, что частотное значение признака строится на равных интервалах ассемблерного кода программ. Градацией признака здесь являются интервалы разбиения и его присутствие в них. Таким образом, F представляет собой набор выделенного числа ассемблерных команд и $F = (f_1, f_2, \ldots, f_n) = (ac_1, ac_2, \ldots, ac_n)$, ac_n – ассемблерная команда $ac_n \in As.com$.

Формирование ЭСП происходит в соответствии со следующими шагами:

1) Каждый файл v_m дизассемблируется и разбивается на интервалы равных длин.

При этом вводится фиксированный множитель для формирования длины шага, корректирующий количество получаемых интервалов k для файлов различного объема, где за длину интервала принимается количество различных ассемблерных команд на промежутке одного шага, k – число интервалов разбиения дизассемблированного кода программы $P_{этал,i}$, где значение признака $L(ac_n) \in \mathbb{N}_0$ и показывает количественное значение для признака ac_n. В исследовании коэффициент k был подобран таким образом, чтобы число интервалов равнялось тридцати.

2) Далее из полученного представления кода выделяется частотная характеристика выбранной градации признака ac_n (одной ассемблерной команды), являющаяся основой для формирования сигнатуры одного исполняемого файла $S'(v_m) = (L(ac_{n,1}), L(ac_{n,2}), \ldots, L(ac_{n,k}))$, где $ac_{n,k}$ – значение частоты появления признака ac_n в k-ом интервале.

3) На основе сигнатур версий v_m программы $P_{этал,i}$ строится промежуточная УСП, представляющая собой $S'_{униф.}(P_{этал,i}) = (p(ac_{n,1}), p(ac_{n,2}), \ldots, p(ac_{n,k}))$, где

$$p(ac_{n,k}) = \begin{cases} \bar{L}(ac_{n,k}), |\min(S'(v_m)[L(ac_{n,k})]) - \bar{L}(ac_{n,k})| \; AND \; |\max(S'(v_m)[L(ac_{n,k})]) - \bar{L}(ac_{n,k})| \le ratio * \bar{L}(ac_n) \\ 0, |\min(S'(v_m)[L(ac_{n,k})]) - \bar{L}(ac_{n,k})| \; OR \; |\max(S'(v_m)[L(ac_{n,k})]) - \bar{L}(ac_{n,k})| > ratio * \bar{L}(ac_n) \end{cases}, \quad (3)$$

где $\bar{L}(ac_{n,k}) = \frac{1}{m}\sum_1^m S'(v_m)[L(ac_{n,k})]$ – среднее значение для $\bar{L}(ac_n)$ всех v_m в $P_{этал,i}$; *ratio* – изменяемый коэффициент; *AND* – логическое «И»; *OR* – логическое «ИЛИ».

Смысл формулы (3) заключается в аннулировании тех значений частот, где абсолютная разница минимального или максимального значения элемента $S'(v_m)[(L(ac_{n,k})]$ от среднего $\bar{L}(ac_{n,k})$ между файлами v_m программы $P_{этал,i}$ в k-ом интервале превышает установленную коэффициентом *ratio* долю от среднего показателя частоты признака в данном интервале для всех версий программы $\bar{L}(ac_{n,k})$.

4) Таким образом, унифицированная сигнатура программы примет вид:

$$S_{униф.}(P_{этал,i}) = (N_i, S'_{униф.}(P_{этал,i})) = (N_i, p(ac_{n,1}), p(ac_{n,2}),\ldots, p(ac_{n,k})).$$

А сигнатура идентифицируемого файла, в соответствии с расчетами под пунктом 2) имеет вид:

$$S(e_j) = (L(ac_{n,1}), L(ac_{n,2}),\ldots, L(ac_{n,k})).$$

3.3.3 Формирование сигнатур программ (СП), используемого в дальнейшем в методе сравнения сигнатур на основе градиентного бустинга деревьев решений

Процесс формирования сигнатур обучающей и тестовой выборки для входных данных для модели обучения градиентного бустинга деревьев решений состоит из тех же этапов, что и подход на основе одной ассемблерной команды, анализируемой на равных интервалах разбиения дизассемблированного кода версий программ. Однако, в данном подходе не происходит формирования унифицированных сигнатур, по причине того, что рассматриваемые алгоритмы лучше работают на большом наборе данных и их классифицирующая способность показала лучшие результаты именно на не унифицированных сигнатурах программ, а на сигнатурах версий этих программ.

Уточним, что сигнатурами файлов обучающей выборки являются структуры вида:

$$S(P_{эталь,i}) = (\{N_i, L(ac_{n,1}), L(ac_{n,2}),\ldots, L(ac_{n,k})\},\ldots, \{N_i, L(ac_{n,1}), L(ac_{n,2}),\ldots, L(ac_{n,k})\}),$$

а СИИФ:

$$S(e_j) = (L(ac_{n,1}), L(ac_{n,2}),\ldots, L(ac_{n,k})).$$

Входными параметрами обучаемых моделей, кроме наименований классов, могут быть лишь количественные меры, однако один из алгоритмов градиентного бустинга деревьев решений способен обрабатывать и категориальные значения.

3.4 Метод сравнения сигнатуры идентифицируемого исполняемого файла (СИИФ) с эталонной сигнатурой программы (ЭСП)

Целью разработки данного метода является повышение точности идентификации исполняемых файлов в условиях:

– мульти-классификации. Каждое наименование программы – это отдельный класс;

– отсутствие класса вредоносных программ как такового. Считается, что программы не обладают ярко выраженными признаками присущими вредоносному ПО. Каждая из программ представляет собой легальный продукт, наличие в АС которого является несанкционированным только в рамках установленных правил действующей политики безопасности;

– ограниченного набора проверяемых программ при проведении аудита. Из приведенного выше материала видно, что АЭСП строится на основе файлов обучающей выборки и распознавание идентифицируемого файла возможно лишь в рамках количества изученных наименований программ;

– постоянного выпуска новых версий ПО разработчиками. Производители поддерживают выпускаемые ими программы, исправляют ошибки, оптимизируют код и добавляют или убирают функционал – это приводит к изменению кода

программ, а с ним и значения выделяемых характеристик. Как уже было отмечено, подходы, основанные на целостности файла, не предоставляют возможности сравнивать две версии одной и той же программы. Присутствует необходимость в постоянном мониторинге программных средств, установленных на АС, и формировании актуальной базы таких показателей.

Из приведенных условий следует потребность в решении задачи по созданию метода сравнения СИИФ с ЭСП, позволяющий производить с высокой точностью идентификацию исполняемых файлов.

Задача сравнения двух сигнатур может рассматриваться, как:

– задача проверки статистической гипотезы о не различности двух распределений (сигнатур) или принадлежности известному закону распределения;

– задача мульти-классификации.

Метод сравнения сигнатур, в зависимости от используемого подхода, может включать в себя этап построения и обучения модели классификатора, а затем определение меры близости того, что идентифицируемый файл является той или иной программой.

3.4.1 Статистические методы сравнения сигнатур на основе критерия однородности хи-квадрат и критерия Колмогорова

Одним из достаточно простых и подходящих в использовании статистических критериев является критерий однородности хи-квадрат, используемый как для дискретных, так и для непрерывных распределений и способный сравнивать распределения характеристик файлов, представленных в любой шкале, начиная от шкалы наименований [83], [84]. Однако, необходимо учитывать ограничение на равное число групп для сравниваемых распределений.

Так, если в результате эксперимента были получены две независимые выборки объемов n_1 и n_2 – представление двух сравниваемых файлов в ассемблерном виде, при этом по рассматриваемому признаку выборки

структурируются в k классов с частотами m_1, m_2, … , m_k и m'_1, m'_2, …, m'_k , то при расчете эмпирического значения χ^2-критерия однородности используют формулу:

$$\chi^2 = n_1 \cdot n_2 \sum_{i=1}^{k} \frac{1}{m_i + m'_i} \cdot \left(\frac{m_i}{n_1} - \frac{m'_i}{n_2} \right)^2$$

где $m_1 + m_2 + \ldots + m_k = n_1$ и $m'_1 + m'_2 + \ldots + m'_k = n_2$.

Доказано, что эта статистика при больших значениях n_1 и n_2 распределена по закону χ^2 с k - 1 степенями свободы [83] и имеет правостороннюю критическую область, поэтому нет оснований отвергнуть гипотезу об однородности распределений если на заданном уровне значимости p выполняется неравенство $\chi^2 < \chi^2_p$.

Стоит заметить, что проверку гипотезы об однородности распределений по χ^2-критерию можно применять не только к двум, но и к нескольким распределениям [84].

Еще одним интересным статистическим критерием является критерий согласия Колмогорова, предназначенного для проверки гипотезы о том, что значения исследуемой выборки подчиняются некоторому известному закону распределения. Процесс идентификации заключается в сравнении сигнатуры программы из АЭСП $S_{униф}(P_{этал,i})$, в которой распределение частот ассемблерной команды является «эталонным» и имеет равномерный закон распределения за счет того, что частота подсчитывалась на интервалах не равных длин, с сигнатурой идентифицируемого файла $S(e_j)$ по критерию согласия Колмогорова.

В отличие от χ^2-критерия, по которому сопоставлялись частоты двух распределений по каждому интервалу разбиения, здесь сопоставление происходит по накопленным частотам, т.е. сначала сопоставляются значения частот встречаемости ассемблерной команды по первому интервалу, затем по сумме первого и второго интервалов, далее по сумме первого, второго и третьего интервалов и т.д. Таким образом, критерий способен найти точку, в которой сумма накопленных расхождений между двумя распределениями является максимальной, и оценить достоверность данного расхождения [84].

Однако критерий Колмогорова-Смирнова, как и все статистические критерии, имеет ряд ограничений, наиболее существенным из которых, в контексте решаемой задачи по идентификации исполняемых файлов, является требование к формированию разрядов значений признака, которые должны быть упорядочены по нарастанию или убыванию, иначе накопление частот будет отражать лишь элемент случайного соседства разрядов [85].

Чтобы учесть данное требование, был разработан особый подход к формированию сигнатур файлов, основанный на формировании интервалов различных длин, но таких, что в каждом из них частота рассматриваемого признака была одинакова и имела вид равномерного распределения.

Статистикой критерия Колмогорова является величина, представляющая собой модуль максимального отклонения эмпирической функции распределения $F^*(x)$ от теоретической $F(x)$, которая при фиксированном объеме выборки n записывается следующим образом [84]:

$$d_n^* = max\ |F^*(x) - F(x)|,\ -\infty < x < \infty$$

Учитывая, что критерий также имеет правостороннюю критическую область, то нет оснований отвергнуть гипотезу о согласии двух распределений, если на уровне значимости p выполняется следующее неравенство $d_n^* < d_{p;n}$.

Отметим также, что при достаточно большом количестве независимых испытаний Колмогоровым была доказана независимость от закона распределения случайной величины, а вероятность $P(\lambda)$ неравенства $\Lambda > \lambda$ приближенно выражается следующим образом и называется функцией обеспеченности [86]:

$$P(\lambda) = P(\Lambda > \lambda) \approx 1 - \sum_{k=-\infty}^{\infty} (-1)^k e^{-2k^2\lambda^2}.$$

Случайная величина $\Lambda = d_n^*\sqrt{n}$ может служить критерием согласия между эмпирическим и теоретическим законами распределения. Если уровень значимости p принять равным 5%, то $\lambda_{кр} \approx 1{,}36$, если 1%, то $\lambda_{кр} \approx 1{,}63$.

3.4.2 Метод сравнения сигнатур на основе машинного обучения

Аппаратная или программная реализация некоторой математической модели, построенной на принципе организации и функционирования биологических нейронных сетей, называется искусственной нейронной сетью, по аналогии с сетями нервных клеток живого организма. Нейронная сеть является частью класса методов искусственного интеллекта и представляет собой обучаемую систему, т.е. ее функционирование происходит не только в соответствии с заданными алгоритмами и функциями, но и на основании полученного опыта.

Структура нейронной сети состоит из нейронов – элементов, представляющих собой простые процессоры и способные быть охарактеризованы каким-либо состоянием, по аналогии с клетками головного мозга. Нейрон обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов. Также нейрон имеет аксон – выходную связь данного нейрона, с которой сигнал поступает на синапсы следующих нейронов [87]. На рисунке 9 показана модель многослойного персептрона, структура которого была выбрана для задачи идентификации ПО экспериментальным путем [73].

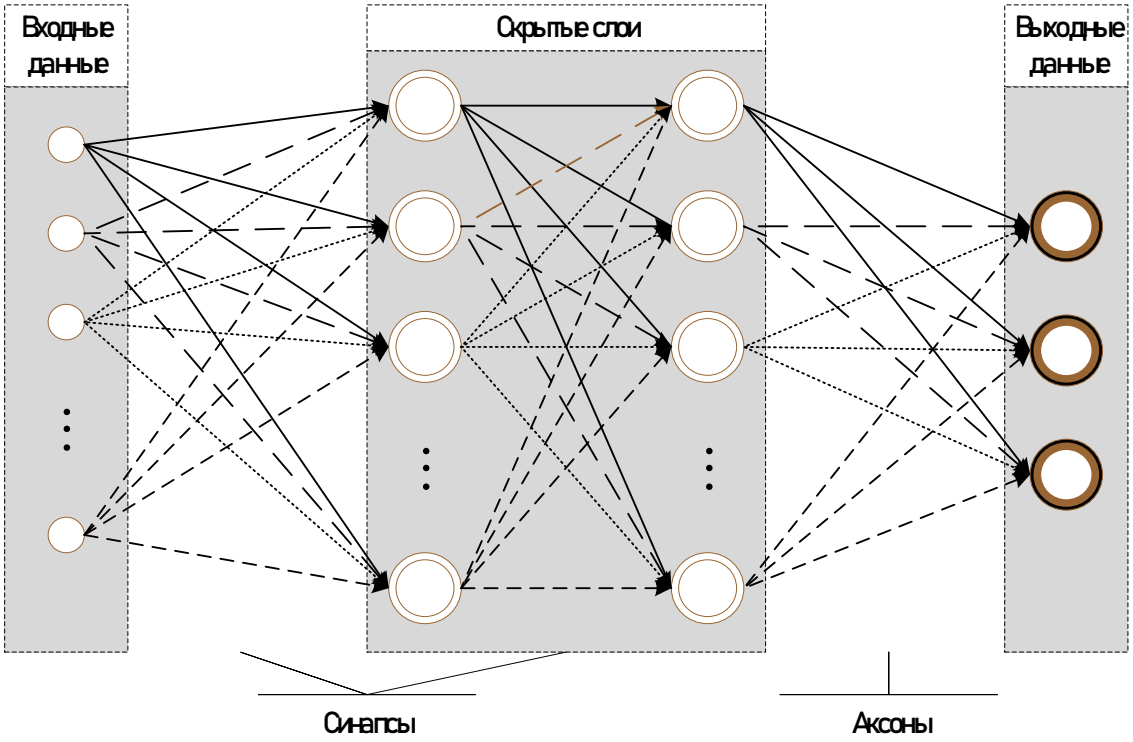


Рисунок 9 – Модель многослойного персептрона с прямой передачей сигнала

Синапсы обладают двумя параметрами – задаваемой величиной синоптической связи x_i и вычисляемым значением веса w_i, который соответствует «силе» одной синоптической связи. Синапс способен оказывать на нейрон как возбуждающее, так и тормозящее воздействие. Именно благодаря весу, происходит изменение входного сигнала при его передаче к другому нейрону. Обучение искусственной нейронной сети происходит путем подачи значений входных переменных во входные элементы модели. Затем последовательно отрабатывают процесс нейроны промежуточных и выходного слоев, где каждый из них вычисляет свое значение активации, беря взвешенную сумму $S=\sum_{i=1}^{n} x_i \cdot w_i$ выходов элементов предыдущего слоя и вычитая из нее пороговое значение w_0, где n – число нейронов предыдущего слоя. Затем значение активации преобразуются с помощью функции активации, и в результате получается выход нейрона. После проведения всех итераций, выходные значения элементов выходного слоя принимаются за выход всей сети в целом [88].

Другим популярным алгоритмом машинного обучения является деревья решений, которые предлагают способ представления правил в иерархической структуре, представляющей собой древовидный граф. Они содержат вершины, в которые записываются проверяемые условия, ребра или ветви, являющиеся переходами из одного узла дерева в другой, а также листья, в которые записываются конечные значения, например, один из классов при решении задачи классификации [89]. На рисунке 10 представлено схематическая модель дерева решений.

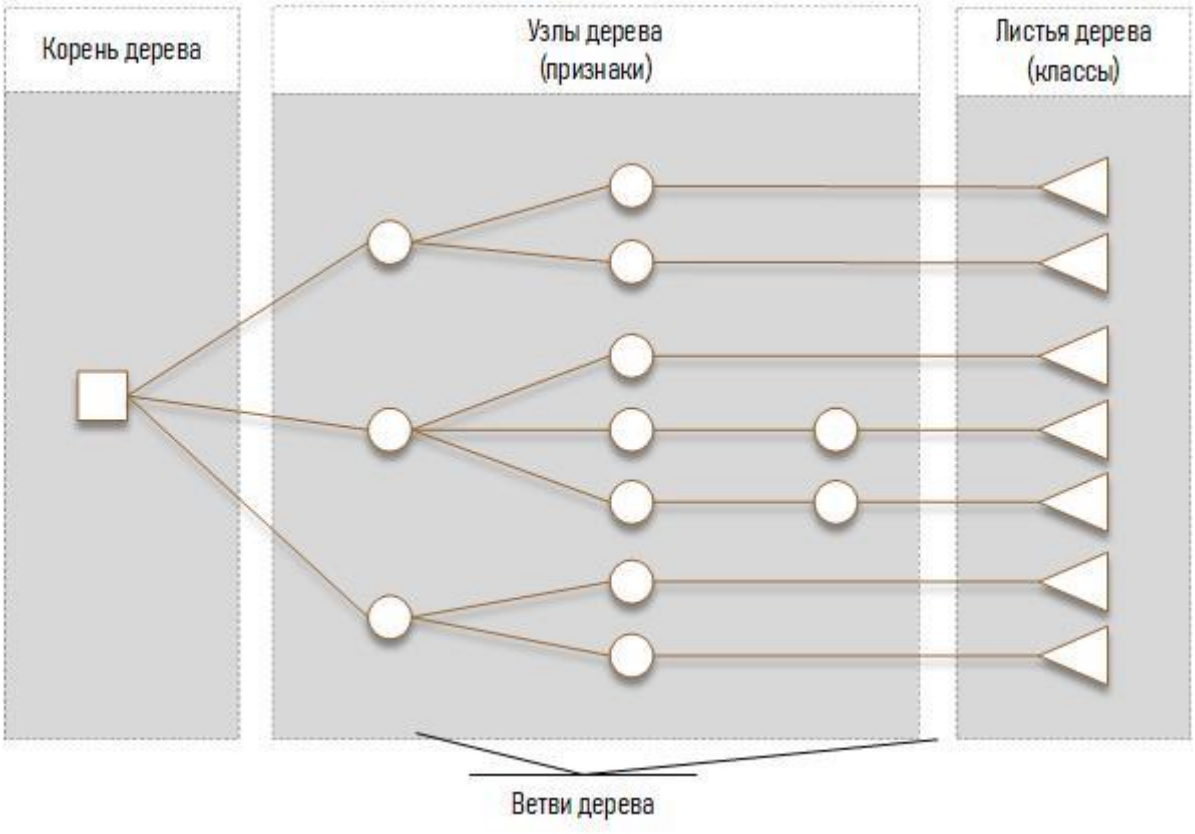


Рисунок 10 – Модель дерева решений

Градиентный бустинг является общим методом для повышения производительности, который может быть использован для любого алгоритма машинного обучения. Суть его алгоритма заключается в обучении каждой последующей модели с использованием данных об ошибках в предыдущих моделях и дальнейшее снижение данных ошибок:

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \rightarrow \min,$$

где x – значения признаков обучаемой выборки; K – число деревьев; f – функция в функциональном пространстве F. В данном исследовании для программы XGBoost это функция Softmax:

$$P(\hat{y} = j|x) = \frac{e^{x^{\mathrm{T}} w_j}}{\sum_{i=1}^n e^{x^{\mathrm{T}} w_i}}$$

коэффициент прогнозирования (prediction scores) для ансамбля деревьев

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

w – весовые коэффициенты признаков; F – набор всех возможных CART (Classification and Regression tree); n – число классов; l – функция потерь обучения (training loss function); Ω – регуляризация (regularization term).

Считается, что градиентный бустинг, можно применить в отношении любого слабого алгоритма в целях снижения его ошибки обучения [90]. Градиентный бустинг деревьев решений позволяет строить аддитивную функцию в виде суммы деревьев решений итерационно, по аналогии с методом градиентного спуска [91]. Так на рисунке 11 схематично представлен ансамбль из k деревьев и принцип минимизации ошибки $obj(\theta)$.

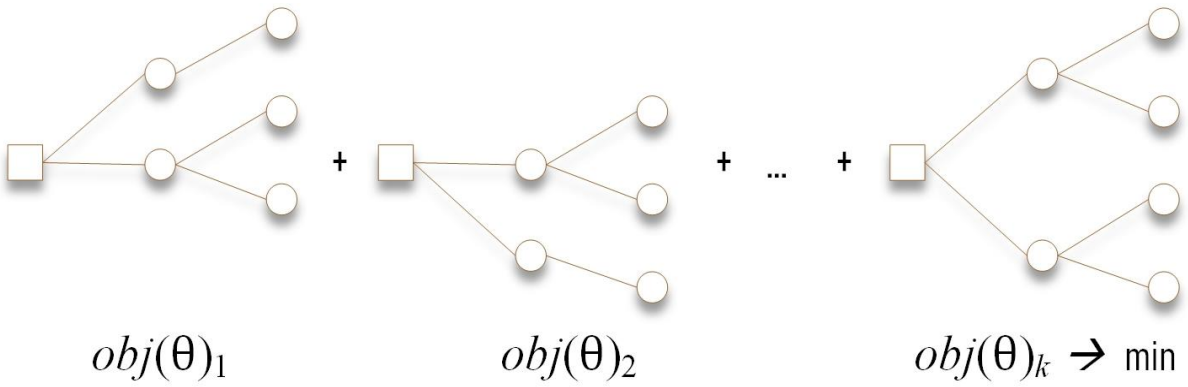


Рисунок 11 – Минимизация ошибки обучения при использовании алгоритма градиентного бустинга деревьев решений

В проведенном исследовании были рассмотрены три программных реализации градиентного бустинга на деревьях решений: LightGBM, XGBoost и CatBoost.

LightGBM – распределенная, быстрая и высокопроизводительная реализация градиентного бустинга, основанная на алгоритмах деревьев решений и применяется для решения задач машинного обучения, таких как ранжирование данных, классификации и многих других. Данная платформа является разработкой компании Microsoft [92].

XGBoost – одна из самых популярных и эффективных реализаций алгоритма градиентного бустинга на деревьях, так же является очень гибкой и портативной [93]. Написанный код работает на большинстве существующих платформ, например, Hadoop, SGE, MPI и способен решать миллиарды примеров [94].

CatBoost – отечественная библиотека, представленная в 2017 г. Особенностями алгоритма являются: построение симметричных деревьев, возможность работы с категориальными признаками, кроме того, он позволяет обучаться на относительно небольшом количестве неоднородных данных [95]. Сравнительный анализ ключевых возможностей представленных реализаций, представлен в таблице 6.

Таблица 6 – Сравнение возможностей описанных платформ

	LightGBM	XGBoost	CatBoost
Скорость обучения модели	высокая	высокая	низкая
Поддержка CPU/GPU	да	да	да
Точность	высокая	высокая	высокая
Поддержка параллельного вычисления	да	да	да
Обработка крупных массивов данных	да	да	да
Поддерживаемые языки программирования	Python, R	Comand Line, Python, R, Julia, JVM языки	Comand Line, Python, R

Стоит отметить, что недостатком всех классифицирующих методов является отсутствие уникального класса, типа «не похоже ни на один из классов», т.е. идентифицируемая сигнатура всегда будет принадлежать одному из классов, сформированных на обучающей выборке. Однако можно произвести дополнительную проверку имени предсказанного класса и имени исследуемого файла на электронном носителе информации. В случае их несовпадения, проверяющему стоит задуматься о наличии нелегитимной программы на исследуемом объекте.

3.4.3 Оценка вычислительной сложности различных методов сравнения СИИФ с ЭСП

Для сравнения вычислительных сложностей двух описанных подходов – на основе статистических критериев и на основе машинного обучения, обратимся к тем этапам построения ЭСП и их сравнения с СИИФ, которые различаются. Так, статистический подход, в отличие машинного, дополняется этапом создания унифицированных сигнатур, который включает в себя этап кластеризации с применением метода *k*-средних и формирования для каждого кластера единого варианта ЭСП. Таким образом, при создании АЭСП вычислительная сложность формирования УСП составляет O(*nkl*), где *n* – число объектов, *k* – число кластеров, *l* – число итераций. Метод же на основе машинного обучения не требует создания УСП, отсюда можно сказать, что вычислительная сложность данного этапа постоянна O(1) [96].

Далее, при рассмотрении этапа непосредственной идентификации исполняемых файлов, для подхода, основанного на статистических критериях, вычислительная сложность попарного сравнения всех сигнатур тестовой выборки с сигнатурами в АЭСП составляет O(*n'*), где *n'* – число объектов (унифицированных ЭСП) в архиве сигнатур, получившихся после этапа кластеризации, где очевидно следующее соотношение $n \geq n' \geq m$ (*m* – количество различных программ). Для подхода же на основе применения алгоритмов

машинного обучения – O(n), где n остается неизменным в виду отсутствия этапа унификации сигнатур.

Из проведенного анализа следует, что для подхода на основе идентификации СИИФ с УСП посредством статистических методов, вычислительная сложность составляет – O(nkl)+O(n'), тогда как для подхода идентификации СИИФ с ЭСП на основе машинного обучения – O(1)+O(n).

3.5 Методика идентификации программного обеспечения на основе статических характеристик программного кода файла

На основе разработанных методов формирования и сравнения ЭСП и СИИФ предложена методика идентификации исполняемого файла, схема которой представлена на рисунке 12.

Методика содержит в себе обязательный подготовительный этап, на котором производится формирование АЭСП, включающем в себя ЭСП или унифицированные ЭСП. И этап идентификации исполняемого файла, состоящий из формирования СИИФ, их непосредственного сравнения с ЭСП из АЭСП, постобработку с применением аддитивного критерия и принятие итогового решения о подлинности предъявленного идентификатора.

В качестве решающего правила выступает выбор наибольшей вероятности принадлежности сигнатуры идентифицируемого исполняемого файла к эталонной сигнатуре программы из архива, по десяти ассемблерным командам:

$$Name(e_j) = \max\{p_{e_j}(N_i) = \sum_{n=1}^{10} \lambda_n \cdot p_{e_j}^{ac_n}(N_i)\}.$$

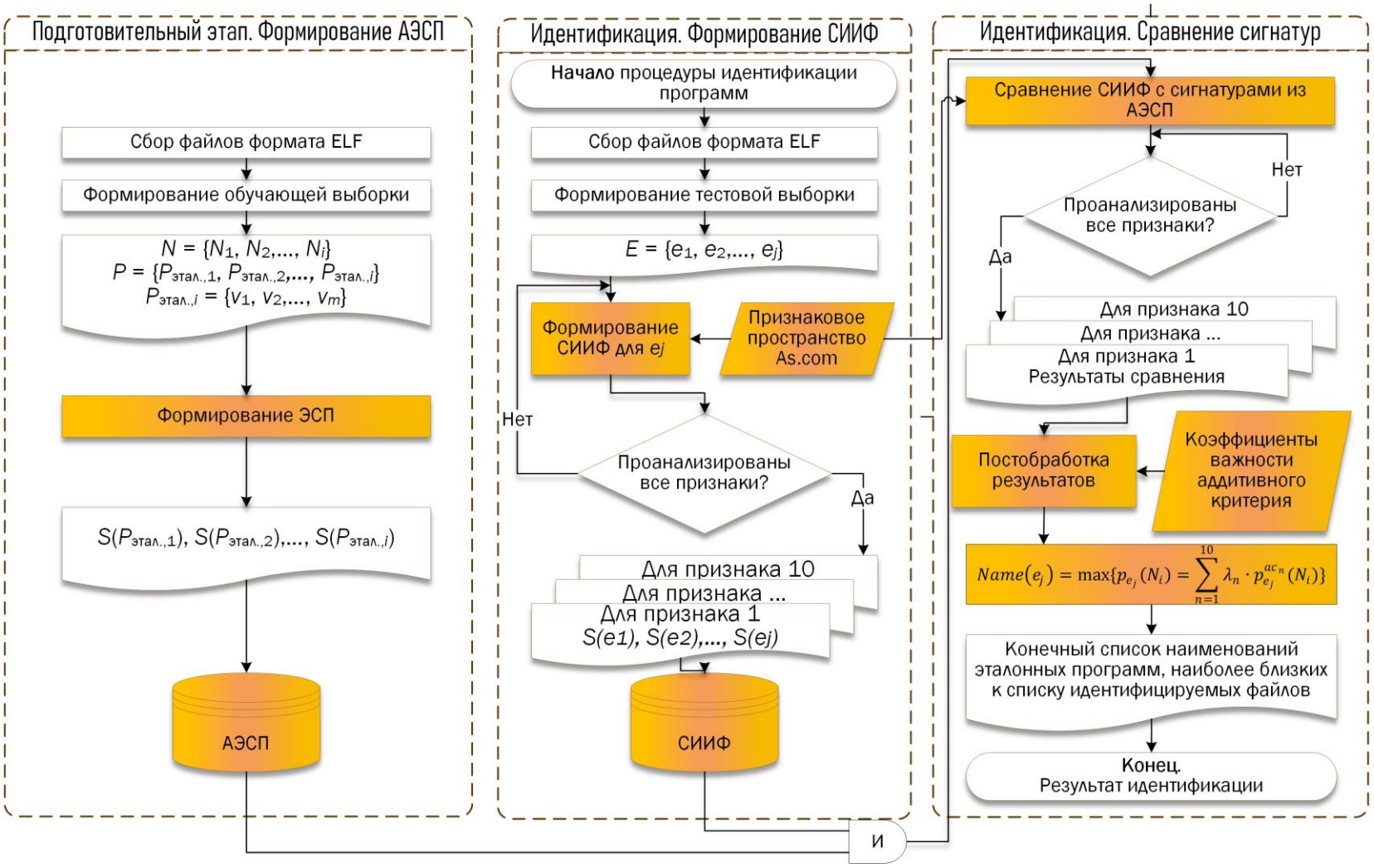


Рисунок 12 – Блок-схема методики идентификации исполняемых файлов

3.5.1 Предварительный этап методики идентификации программного обеспечения. Сбор и формирование обучающей выборки и составление АЭСП, содержащий ЭСП или унифицированные ЭСП

Идентификация исполняемых файлов возможна только при наличии сформированного заранее архива (АЭСП), содержащего в себе ЭСП или унифицированные ЭСП, по причине того, что в процессе идентификации итерационно происходит сравнение последовательности измеренных характеристик идентифицируемого файла с такими же ранее собранными характеристиками версий программного обеспечения, хранящимися в АЭСП, для которых известна их принадлежность к той или иной программе.

Первым шагом является сбор, обработка и структурирование версий программ. На данном этапе формируются множества N и P, а также различные версии $\{v_1, v_2, \ldots, v_m\}$ по каждой $P_{этал,i}$, представляющие собой обучающую выборку для того, чтобы в последствии имелась возможность отнесения идентифицируемого файла к одному из имеющихся наименований программ.

Список всех рассматриваемых программ и их версий обязательно должен быть обновляемым и дополняемым на протяжении всего эксплуатационного периода защищаемой ИС.

Формирование ЭСП происходит путем дизассемблирования файлов обучающей выборки и подсчета частот встречаемости выбранного признака (рисунок 13). Которые в конце данного этапа формируют АЭСП.

Предлагается следующий алгоритм сбора версий программ и сохранения их ЭСП:

1) На основании действующих правил организации формируется эталонный список разрешенного или запрещенного ПО, регулярный пересмотр и обновление которого, является важным требованием.

Формирование такого списка может происходить несколькими путями, например, информация об эталонных программах может быть получена с электронных носителей информации, содержащих только легитимное ПО или

наоборот запрещенное; информацию можно получить, выгрузив дистрибутивы программ с официальных репозиториев ОС Linux и другими подходящими для задачи выделения определенного пула программ способами.

Автоматизация сбора программ легко реализуема путем написания скрипта на каком-либо языке программирования.

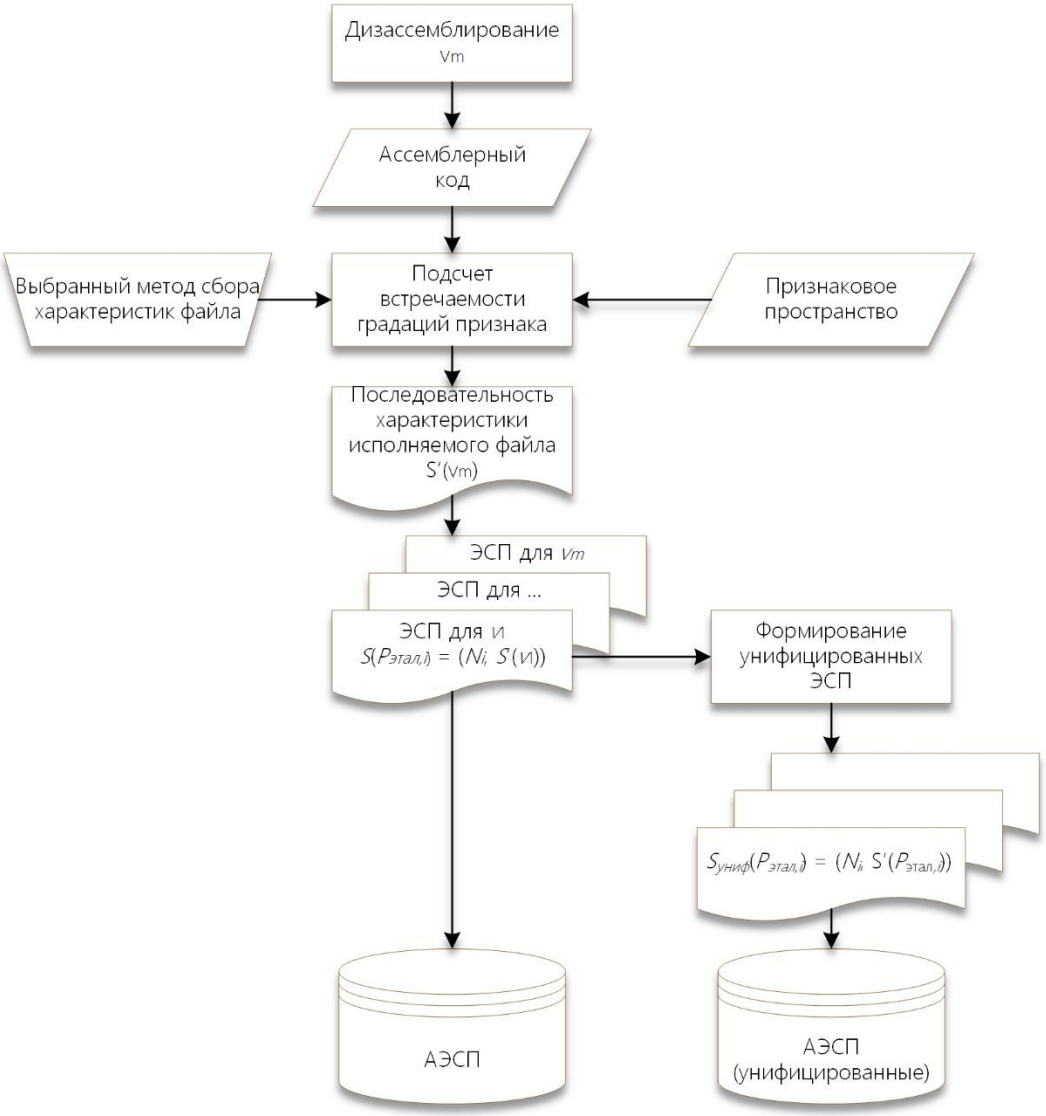


Рисунок 13 – Сбор характеристик файлов

2) После сбора ELF-файлов, необходимо произвести их обработку и структурирование.

Она заключается в формировании двух множеств: непосредственно программ $P = \{P_{\text{этал.},1}, P_{\text{этал.},2}, \ldots, P_{\text{этал.},i}\}$ с их различными версиями $P_{\text{этал.},i} = \{v_1, v_2, \ldots, v_m\}$ и множества имен этих программ $N = \{N_1, N_2, \ldots, N_i\}$.

3) Следующий шаг состоит в формировании СП, для чего происходит дизассемблирование каждого ELF-файла и выделение из всего полученного кода последовательностей ассемблерных команд.

В зависимости от выбранного метода сбора характеристик файла и способа построения сигнатур на основе признакового пространства, производится подсчет встречаемости градации признака (ассемблерной команды) по каждому ELF-файлу в отдельности. После чего формируются промежуточные сигнатуры версий программ $S'(v_m)$ и, на их основе, создаются СП $S(P_{этал,i})$ сохраняемые в АЭСП. При этом, для методов идентификации, основанных на статистических критериях, производится дополнительный шаг, заключающийся в кластеризации СП $S(P_{этал,i})$ и формировании унифицированных ЭСП, записываемых в АЭСП унифицированных сигнатур.

Стоит отметить, что в данном исследовании были выбраны десять ассемблерных команд в качестве признаков (для тех методов формирования сигнатур, которые основаны на подсчете частоты встречаемости градации признака на интервалах ассемблерного кода программ), анализ которых в совокупности позволяет достичь белее высоких показателей точности идентификации, чем их использование по отдельности или всех вместе за раз. Использование аддитивного критерия Фишберна будет приведено в пункте 3.2.3.

3.5.2 Основные этапы методики идентификации исполняемого файла

Идентификация ELF-файлов при проведении мероприятий по аудиту ПО на электронных носителях информации состоит из двух подэтапов: формирования СИИФ исследуемых файлов и процесса сравнения сигнатур и присвоения идентификатора (имени программы).

Алгоритм формирования СИИФ состоит в следующем:

1) Как и в предварительном этапе методики производится автоматический сбор всех ELF-файлов с анализируемого электронного носителя информации.

2) Формирование СИИФ происходит путем дизассемблирования файлов тестовой выборки и подсчета частот встречаемости выбранного признака (также, как и на рисунке 12).

Для каждого идентифицируемого файла выполняется итерационный процесс формирования СИИФ по всем десяти ассемблерным командам в точности, как и при формировании ЭСП. После чего все СИИФ формируют некоторую базу исследуемых сигнатур.

Алгоритм сравнения СИИФ с ЭСП из АЭСП состоит в следующем:

1) В зависимости от выбранного метода идентификации: при помощи статистического критерия или машинного обучения, происходит не только выбор способа формирования сигнатур, произведенный ранее в методике, но и, в соответствии с ним, способ сравнения сигнатур, а также выбор АЭСП.

2) Процесс сравнения сигнатур происходит путем попарного сравнения двух последовательностей признака – у идентифицируемого файла и у эталонной программы.

Так при формировании унифицированных ЭСП и сравнении их при помощи статистического критерия или нейронной сети, необходимо обратиться к АЭСП (унифицированных), при формировании ЭСП и сравнении их при помощи градиентного бустинга, необходимо обратиться к АЭСП.

Сравнение сигнатур для идентифицируемого файла также происходит по всем десяти ассемблерным командам, после чего следует этап постобработки результатов.

3) Постобработка результатов состоит в формировании конечного решения о принадлежности идентифицируемого файла к той или иной программе и заключается в применении коэффициентов важности аддитивного критерия Фишберна к результатам идентификации по десяти ассемблерным командам.

Коэффициенты должны быть рассчитаны заранее. Автором данные коэффициенты были рассчитаны при проведении эксперимента на результатах идентификации тестовой выборки.

В итоге, после произведения сравнения всех СИИФ с ЭСП, получается список имен программ, находящихся на исследуемом электронном носителе информации.

3.6 Оценка точности идентификации программного обеспечения

Оценка качества или эффективности методики, а также разработанных методов в частности, подразумевает оценку точности идентификации исполняемых файлов. В исследовании было использовано три подхода к оценке классификации: матрица ошибок (*Confusion Matrix*), точность (*Accuracy*), бикубическая мера (*F-measure*).

Оценка точности идентификации, на основе использования статистических критериев, оценивалась при помощи матрицы ошибок, позволяющей рассмотреть все возможные результаты, ее структура приведена в таблице 7.

Таблица 7 – Матрица ошибок (*Confusion Matrix*)

Генеральная совокупность (Total population)		Реальное состояние (True condition)	
		Положительное состояние (Condition positive)	Отрицательное состояние (Condition negative)
Прогнозируемое состояние (Predicted condition)	Прогнозируемое состояние положительное (Predicted condition positive)	Верно положительное (True positive)	Не верно положительное (False positive)
	Прогнозируемое состояние отрицательное (Predicted condition negative)	Не верно отрицательное (False negative)	Верно отрицательное (True negative)

За точность идентификации *Accuracy* выступает отношение числа верно идентифицированных исполняемых файлов *TP* (true positive results) к общему объему исполняемых файлов, участвовавших в идентификации *J* (числу файлов

тестовой выборки). Показатель *Accuracy* считается по данной формуле и представляется в процентах:

$$Accuracy = \frac{TP}{J} \cdot 100\%.$$

Точность классификатора можно рассчитать при помощи *F-measure*, которая представляет собой среднее гармоничное между точностью (precision) и полнотой (recall). Так для мульти-классификатора необходимо произвести расчет *F-measure* по каждому объекту (программы тестовой выборки) по формуле:

$$Fmeasure(P_{\text{тест.},i}) = \frac{1}{\alpha \cdot \frac{1}{precision} + (1-\alpha) \cdot \frac{1}{recall}},$$

где коэффициент α позволяет взвешивать соотношение точности (precision) и полноты (recall), и принимает значение от нуля до единицы включительно.

Бикубическая мера для всех объектов рассчитывается как среднее значение *F-measure* для каждого из *i* объектов:

$$Fmeasure = \frac{\sum_1^i Fmeasure(P_{\text{тест.},i})}{i}.$$

Разнообразие в подходе оценки точности вызвано следующими факторами:

– для учета всех результатов в подходе идентификации на основе статистических критериев необходимо рассматривать матрицу ошибок (таблица 7), которая включает в себя все возможные результаты, получаемые при проведении статистического анализа;

– для учета всех результатов в подходе идентификации на основе машинного обучения, применительно к задаче мульти-классификации, матрица ошибок уже не подходит, т.к. на выходе присутствуют результаты только двух видов – true positive (файл *верно* идентифицируется как программа) и false positive (файл *не верно* идентифицируется как программа), таким образом для оценки точности классификатора подходит вычисление *Accuracy*;

– оценка бикубической меры производилась в целях сравнения получаемых результатов точности с существующими методами идентификации других исследователей.

3.7 Ограничения методики

Важно понимать, что методике идентификации ПО присущи ограничения, посредством влияния ряда факторов, таких как:

– потенциальное количество различных программ, установленных на одном компьютере или на всех компьютерах организации;

– количество различных версий для каждой программы, измеряемое от одной до десятков, но в практике не превосходящее сотни;

– наличие существенных изменений в различных версиях одной программы;

– наличие индивидуального характера распределения признака идентифицируемого файла от других программ;

– возможность производить дизассемблирование исполняемого файла;

– не способность в выявлении закладки вредоносного кода в легитимный исполняемый файл;

– классификация не способна выдать корректный результат для программы, класс которой не был определен на этапе формирование модели классификации (нельзя создать класс «файл не похож ни на одну из программ»);

– и многих других.

Выводы по главе 3

а) В третей главе был произведен анализ структуры ELF-файла, его представление в ассемблерном коде и выявлены характеристики пригодные для использования при формировании сигнатур исполняемых файлов.

б) Представлен подход к формированию признакового пространства с выделением наиболее информативных признаков и их дальнейшего использования.

в) Сформирована модель представления программного обеспечения, заключающая в себе наименования и программные реализации кодов, версии программ и их внутренние характеристики, представляющие собой последовательность появления выбранного признака.

г) Разработаны методы по формированию ЭСП и унифицированных ЭСП, создание АЭСП, на основе которого происходит дальнейшая идентификация исполняемых файлов.

д) Предложено несколько методов по проведению сравнения нескольких сигнатур ЭСП и СИИФ, основанных как на проведении статистического анализа с применением критерия однородности хи-квадрат и критерием согласия Колмогорова, так и с использованием алгоритмов машинного обучения, в частности искусственной нейронной сети и градиентного бустинга деревьев решений.

е) Представлен анализ вычислительной сложности методов сравнения сигнатур. Так подход на основе машинного обучения является менее затратным по вычислительным ресурсам и простым в исполнении.

ж) На основе разработанных методов формирования сигнатур и их сравнения была предложена методика идентификации исполняемых файлов.

з) Необходимо произвести ряд экспериментов, определяющих точность идентификации файлов с помощью предложенной методики. А также анализ методов, описанных в ней и выделение наиболее эффективного с точки зрения качества классификации.

Глава 4. Экспериментальная проверка точности идентификации программного обеспечения и анализ полученных результатов

В данной главе описывается серия проведенных экспериментов, целью которых являлась проверка достигаемой точности идентификации исполняемых файлов, получаемых при использовании разработанной методики, а также выявление наиболее результативного метода по формированию и сравнению сигнатур ELF-файлов.

Эксперименты производились на файлах, скаченных с официального репозитория Linux в 2016 году, формирующих из собой базу из 63 программ, 578 исполняемых файлов, более детальное описание было описано в разделе 2.2. Отсюда соотношение обучающей к тестовой выборке составляет 79% к 21%.

4.1 Входные данные экспериментов. Обучающая и тестовая выборки

База программ была разделена на две выборки: обучающую с 455 файлами и тестовую со 123 файлами. По разработанной методике идентификации, файлы были предварительно разделены по версиям различных программ, для обучающей выборки из них были выделены исследуемые характеристики, формирующие ЭСП или унифицированные ЭСП, и помещены в АЭСП.

Помимо того, что одной из ключевых особенностей рассматриваемой задачи является ограниченное число идентифицируемых файлов, с этим связана еще проблема ограниченного числа версий по каждой программе. Реализуемая методика идентификации должна позволять производить идентификацию исполняемых файлов в условиях дисбаланса числа версий для различных программ.

Для приближения экспериментов к реальной ситуации, обучающая выборка была составлена таким образом, что для различных программ число используемых различных версий для них было неравномерно.

В представленных ниже экспериментах, оценка точности производилась в соответствии с описанными критериями точности в разделе 2.6.

4.2 Определение точности идентификации при использовании разработанных методов сравнения СИИФ с ЭСП

Цель эксперимента – оценка сравнительного анализа точности идентификации ПО с использованием разработанных методов по формированию сигнатур исполняемых файлов и их сравнение с различными существующими методами, а также с показателями точности идентификации, полученными авторами других работ.

В ходе теоретического исследования были разработаны несколько подходов к сбору характеристик исполняемых файлов и их сравнению, чьи показатели точности были исследованы ниже.

В таблицах 8-10, 12, 14 и на рисунках 14-16, 18, 19, 22 представлены полученные результаты. А также, для наглядности, рассмотрим на примерах описанные ранее в разделе 2.4 методы сравнения СИИФ с ЭСП.

4.2.1 Точность идентификации при использовании статистических методов сравнения сигнатур

Подход к построению унифицированных ЭСП и СИИФ на основе распределения 118 ассемблерных команд и идентификации при помощи критерия однородности хи-квадрат.

Пусть идентифицируемым файлом является программа *bacula-console* версии *5.2.5-0ubuntu6*. Сравнение выбранного файла будет происходить с программами: *bacula-console* и *baobab*. Таким образом, были сформированы сигнатура идентифицируемого файла и заранее на обучающей выборке две сигнатуры выбранных программ.

На рисунках 14 и 15 показаны распределения частот признака в сигнатуре идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и соответственно в сигнатурах программ *bacula-console* и *baobab* из АЭСП. По оси абсцисс отложены градации признака с ненулевыми значениями в сигнатуре, по оси ординат – частоты ненулевых ассемблерных команд из сигнатуры программы.

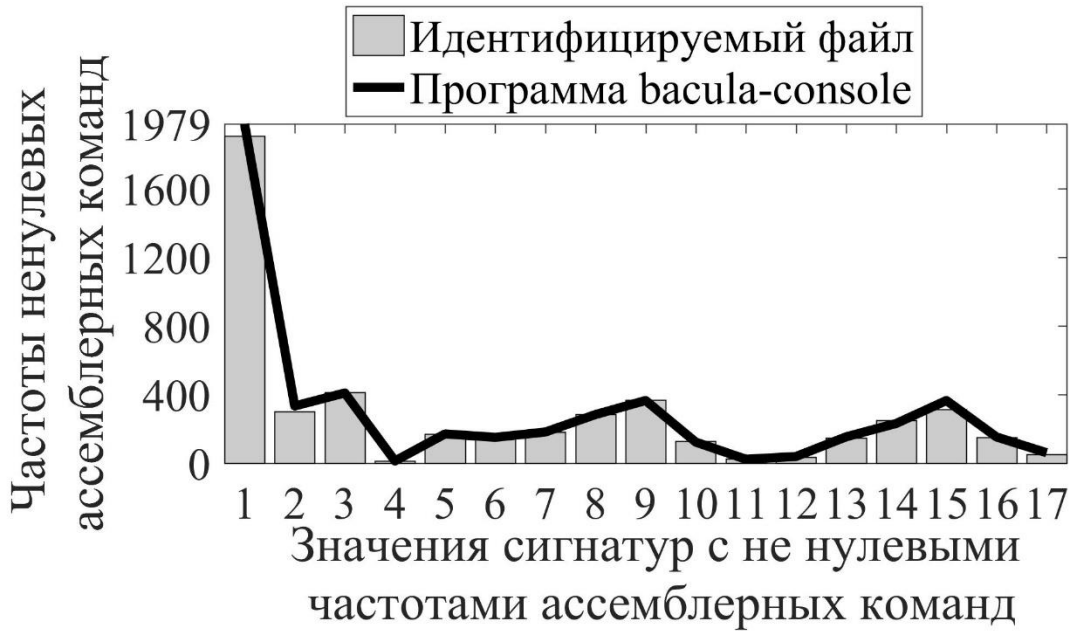


Рисунок 14 – Сигнатуры идентифицируемого файла является *bacula-console_5.2.5-0ubuntu6* и программы *bacula-console*

При проверке статистической гипотезы о сходстве распределения частот градаций признака в сигнатуре идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и в сигнатуре программы *bacula-console_i386* из АЭСП эмпирическое значение критерия однородности $\chi^2 = 6{,}76$, что ниже критического значения $\chi^2_p = 26{,}30$, поэтому на уровне значимости $p = 0{,}05$ можно утверждать, что идентифицируемый файл является программой из архива (что действительно так).

Действительно, на рисунке 14 видно, что гистограмма частот встречаемости градаций признака идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и кривая распределения частот для программы *bacula-console* различаются незначительно.

В свою очередь при проверке статистической гипотезы о сходстве распределения частот градаций признака в сигнатуре идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и в сигнатуре программы *baobab_i386* из АЭСП получено эмпирическое значение критерия однородности $\chi^2 = 301{,}903$, превышающее критическое значение $\chi^2_p = 35{,}17$ на уровне значимости $p = 0{,}05$. Следовательно, на данном уровне значимости можно утверждать, что

идентифицируемый файл не является программой из архива (что действительно так).

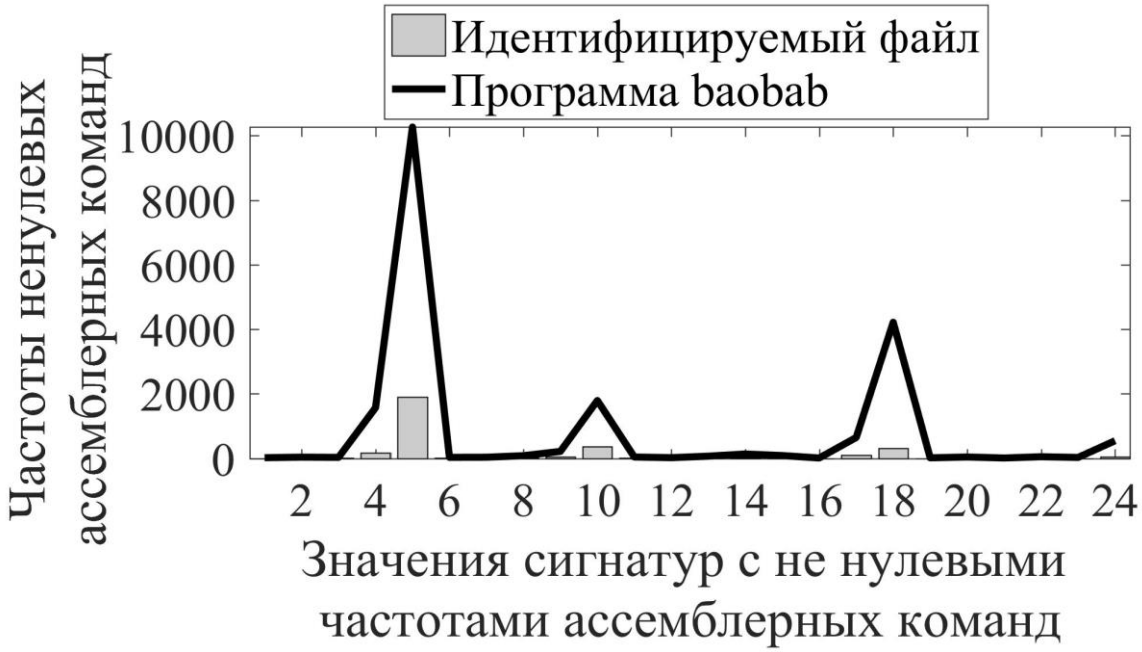


Рисунок 15 – Сигнатуры идентифицируемого файла является *bacula-console_5.2.5-0ubuntu6* и программы *baobab*

Полученный вывод подтверждается на рисунке 15, где графически показано, что различия в распределениях частот встречаемости градаций признака идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и программы *baobab_i386* из архива значительны.

Общие результаты эксперимента с применением критерия однородности хи-квадрат для распределения 118 ассемблерных команд представлены в матрице ошибок (таблица 8), где в первой и четвертой строках содержится показатель (в процентах) корректных результатов, в третьей и четверной строках – показатель ошибки первого и второго рода соответственно.

В первом и втором столбцах таблицы описываются все возможные исходы идентификации сигнатур, а в третьем столбце отображены полученные результаты идентификации для этих исходов, выраженные в процентных долях.

Таблица 8 – Матрица ошибок для подхода на основе критерия однородности хи-квадрат

Отношение между ЭСП из АЭСП и СИИФ	Основная гипотеза о сходстве сигнатур	Результат эксперимента, %	
		Уровень значимости $p = 0,05$	Уровень значимости $p = 0,01$
Сигнатуры относятся к одной и той же программе	Принимается	0,36	0,42
	Отвергается	1,23	1,17
Сигнатуры относятся к разным программам	Принимается	0,10	0,16
	Отвергается	98,32	98,25

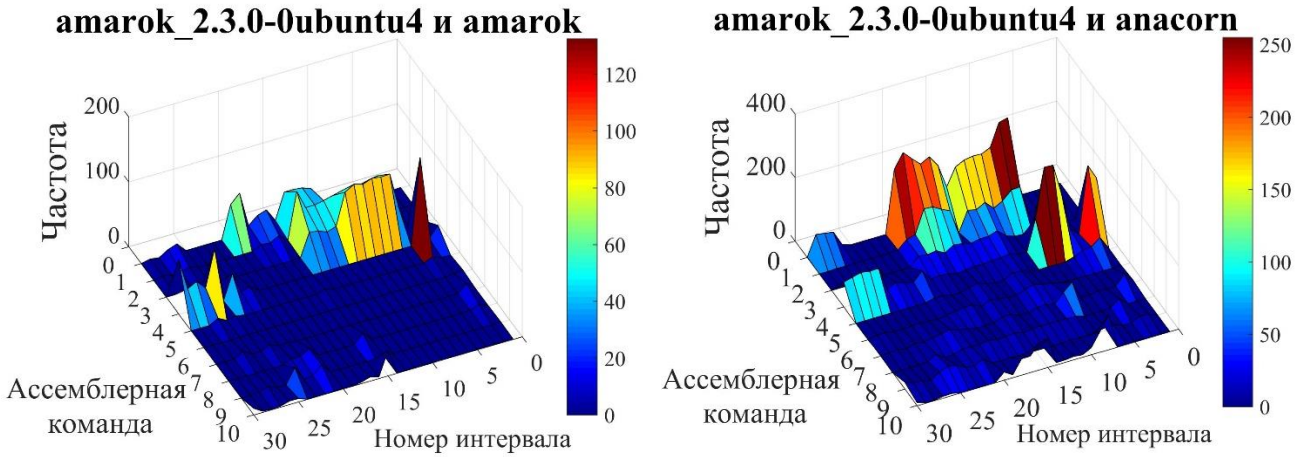
Из таблицы 8 следует, что показатель точности на основе *Confusion Matrix* для идентификации сигнатур ELF-файлов, построенных на основе частот 118 ассемблерных команд, составляет 98,68% для уровня значимости $p = 0,05$ и 98,67% для $p = 0,01$.

Подход к построению унифицированных ЭСП и СИИФ на основе распределения одной ассемблерной команды на равных интервалах и идентификации при помощи критерия однородности хи-квадрат.

Пусть идентифицируемым файлом является программа *amarok* версии *2.3.0-0ubuntu4*. Сравнение выбранного файла будет происходить с программами: *amarok* и *anacorn*. Таким образом, были сформированы сигнатура идентифицируемого файла и заранее на обучающей выборке две сигнатуры выбранных программ.

На рисунке 16 изображены контурные графики, где для построения трехмерной поверхности использовалась абсолютная разница между десятью распределениями ассемблерных команд (сигнатурами СИИФ и ЭСП), построенными для идентифицируемого файла *amarok_2.3.0-0ubuntu4* и десятью унифицированными сигнатурами из АЭСП для двух программ *anacorn* и *amarok*.

По оси абсцисс отмечены номера интервалов разбиения ассемблерной последовательности эталонной программы k от 1 до 30, а по оси ординат – порядковый номер ассемблерной команды, используемой для построения сигнатур.



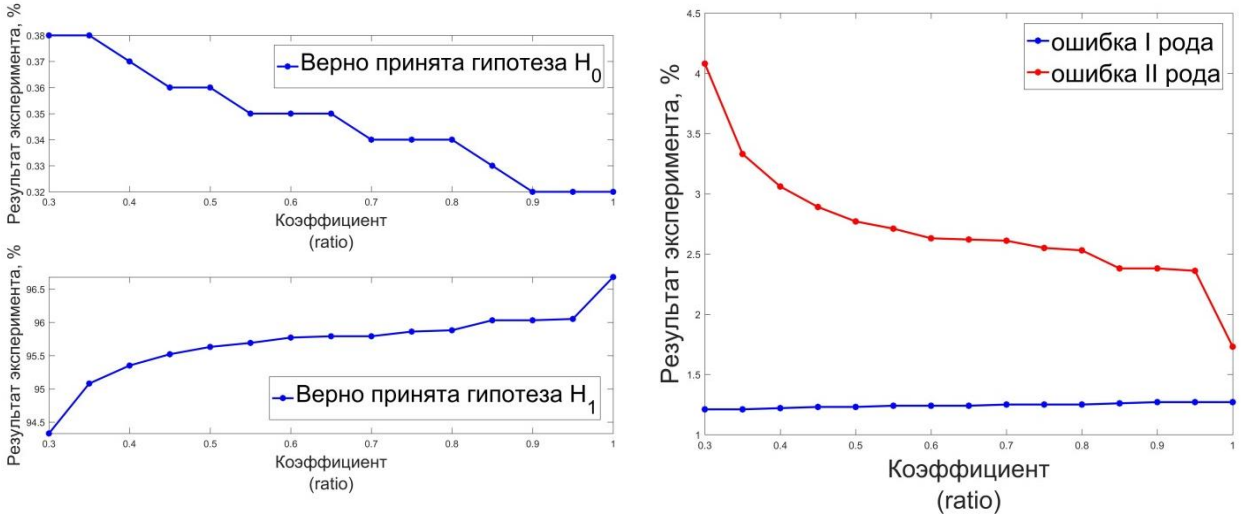
а) б)

Рисунок 16 – Абсолютная разница между СИИФ *amarok_2.3.0-0ubuntu4* и унифицированных ЭСП а) *amarok* (левый) и б) *anacorn* (правый)

Видно, что для сигнатур, относящихся к одной и той же программе (рисунок 16 а)), высота пиков не превышает значения 150, а для многих ассемблерных команд и вовсе минимальна, у различных же программ (рисунок 16 б)) поверхность имеет большее число возвышений, а их максимальное значение почти в два раза больше, чем для одинаковых программ.

На рисунке 17 представлено влияние изменения значения коэффициента (*ratio*) в формуле (3) на результат идентификации. Эксперимент проводился для ассемблерной команды *call*, а значение коэффициента (*ratio*) изменялось от 0,3 до 1 с шагом 0,05.

На графиках рисунка 17 а) изображены корректные результаты идентификации, т.е. когда верно принята основная гипотеза H_0 (верхний) или верно принята альтернативная гипотеза H_1 (нижний). На графике рисунка 17 б) изображены результаты возникающих ошибок первого и второго рода, когда неверно отвергнута основная гипотеза H_0 и когда неверно принята основная гипотеза H_0.



а) б)

Рисунок 17 – Зависимость результатов идентификации от коэффициента (*ratio*)

Как видно, оптимальным значением коэффициента (*ratio*) является 0,6, именно в этой точке график ошибки второго рода замедляет свой рост.

Общие результаты эксперимента с применением критерия однородности хи-квадрат для распределения одной из десяти ассемблерным командам на равных интервалах представлены в матрице ошибок (таблица 9). Принятие или отклонение основной гипотезы H_0 принималось на уровне значимости p = 0,05.

Таблица 9 – Матрица ошибок для подхода на основе критерия однородности хи-квадрат

Отношение между ЭСП из АЭСП и СИИФ			Сигнатуры относятся к одной и той же программе		Сигнатуры относятся к разным программам	
Основная гипотеза о сходстве сигнатур			Принимается	Отвергается	Принимается	Отвергается
Результат эксперимента, %	Ассемблерная команда	add	0,16	1,44	0,01	98,4
		and	0,3	1,29	1,37	97,04
		call	0,34	1,25	2,19	96,22
		cmp	0,44	1,15	6,34	92,07
		je	0,45	1,14	3,43	94,98
		jmp	0,33	1,26	1,15	97,26
		lea	0,47	1,12	11,44	86,97
		mov	0,22	1,37	0,69	97,71
		pop	0,33	1,26	2,34	96,07
		push	0,26	1,33	0,74	97,67

Из таблицы 9 следует, что наибольший показатель точности на основе *Confusion Matrix* для идентификации сигнатур ELF-файлов, построенных на основе частот выбранной ассемблерной команды, составляет 98,56% для ассемблерной команды *add*.

Подход к построению унифицированных ЭСП и СИИФ на основе распределения одной ассемблерной команды на неравных интервалах и идентификации при помощи критерия согласия Колмогорова.

В качестве признака для формирования частотных распределений была выбрана ассемблерная команда *cmp*.

Пусть идентифицируемым файлом является программа *bacula-console* версии *5.2.5-0ubuntu6*. Сравнение выбранного файла будет происходить с программами: *bacula-console* и *baobab*. Таким образом, были сформированы сигнатура идентифицируемого файла и заранее на обучающей выборке две сигнатуры выбранных программ.

На рисунках 18 и 19 изображены гистограммы накоплений относительных частот для сигнатуры идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и кривые соответственно для сигнатур программ *bacula-console* и *baobab* из архива. По оси абсцисс отложена последовательность интервалов разбиения дизассемблированного кода программ с ненулевой частотой встречаемости признака, по оси ординат – накопления частот встречаемости ассемблерной команды *cmp* на этих интервалах.

Как видно из рисунка 18, накопление относительных частот в сигнатуре идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* незначительно отклоняется от накопления относительных частот в сигнатуре программы *bacula-console* и накопления для равномерного закона распределения. Максимальное расхождение составляет $d_n^* = 0{,}073$, которое меньше критического значения $d_{p;n} = 0{,}106$, поэтому на уровне значимости $p = 0{,}05$ можно утверждать, что идентифицируемый файл является программой из АЭСП (что действительно так).

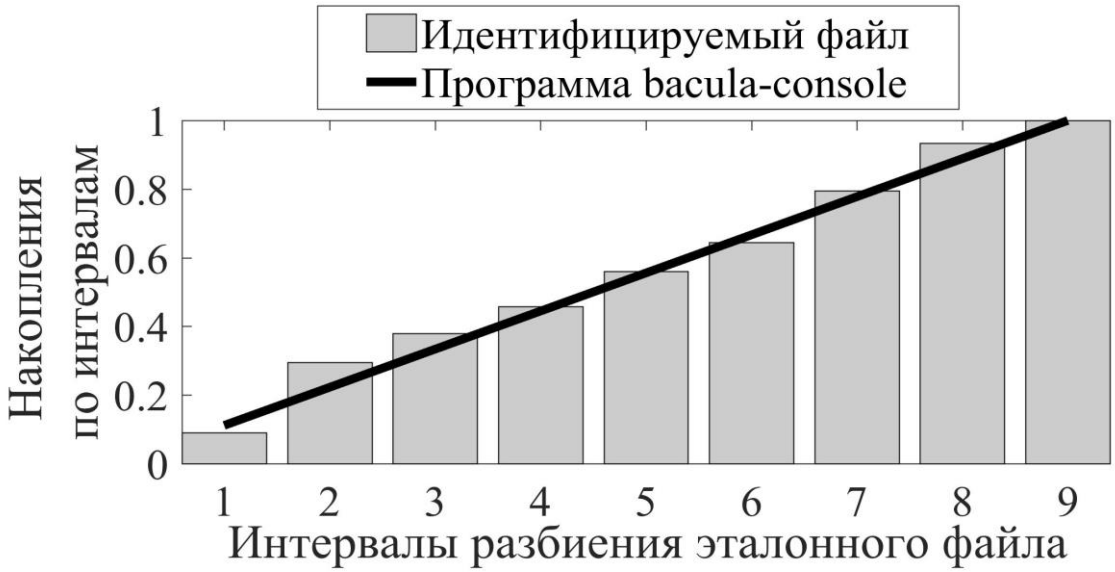


Рисунок 18 – Сигнатуры идентифицируемого файла *является bacula-console_5.2.5-0ubuntu6* и программы *bacula-console*

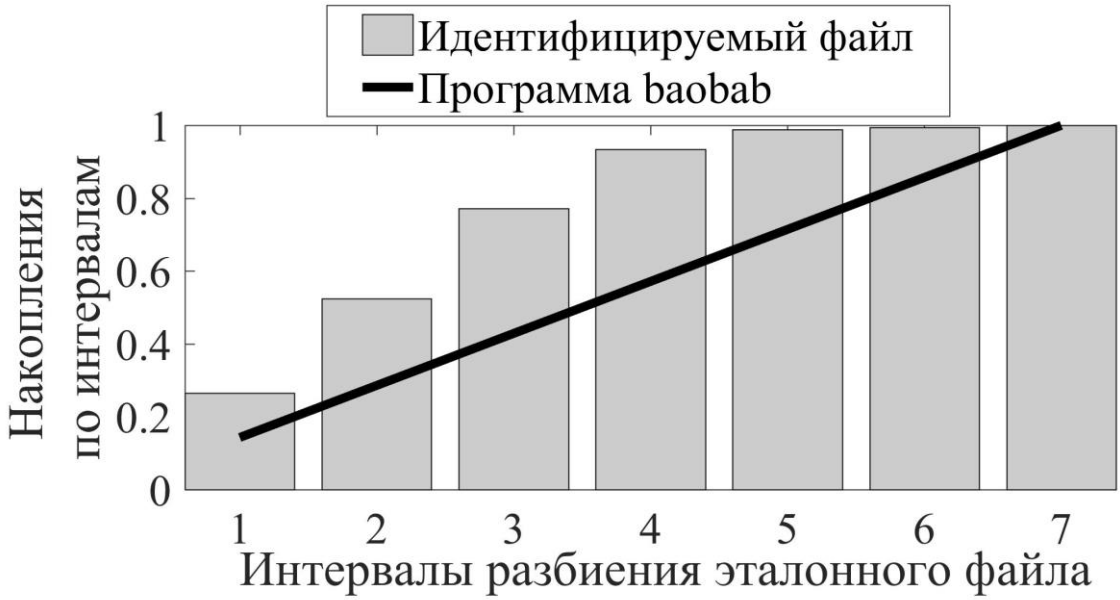


Рисунок 19 – Сигнатуры идентифицируемого файла является *bacula-console_5.2.5-0ubuntu6* и программы *baobab*

Из рисунка 19 следует, что расхождения между накоплениями относительных частот в сигнатуре идентифицируемого файла *bacula-console_5.2.5-0ubuntu6* и в сигнатуре программы *baobab_i386* значительны, при этом максимальное расхождение составляет $d_n^* = 0{,}362$, которое больше критического значения $d_{p;n} = 0{,}106$, поэтому на уровне значимости $p = 0{,}05$ можно утверждать,

что идентифицируемый файл не является программой из АЭСП (что действительно так).

Общие результаты эксперимента с применением критерия согласия Колмогорова представлены в матрице ошибок (таблица 10).

Таблица 10 – Матрица ошибок для подхода на основе критерия согласия Колмогорова

Отношение между ЭСП из АЭСП и СИИФ	Основная гипотеза о сходстве сигнатур	Результат эксперимента, %	
		Уровень значимости $p = 0,05$	Уровень значимости $p = 0,01$
Сигнатуры относятся к одной и той же программе	Принимается	1,86	1,94
	Отвергается	0,61	0,53
Сигнатуры относятся к разным программам	Принимается	4,30	6,39
	Отвергается	93,23	91,14

Из таблицы 10 следует, что показатель *Accuracy* на основе *Confusion Matrix* для идентификации сигнатур ELF-файлов, построенных на основе частот выбранной ассемблерной команды, составляет 95,09% для уровня значимости $p = 0,05$ и 93,08% для $p = 0,01$.

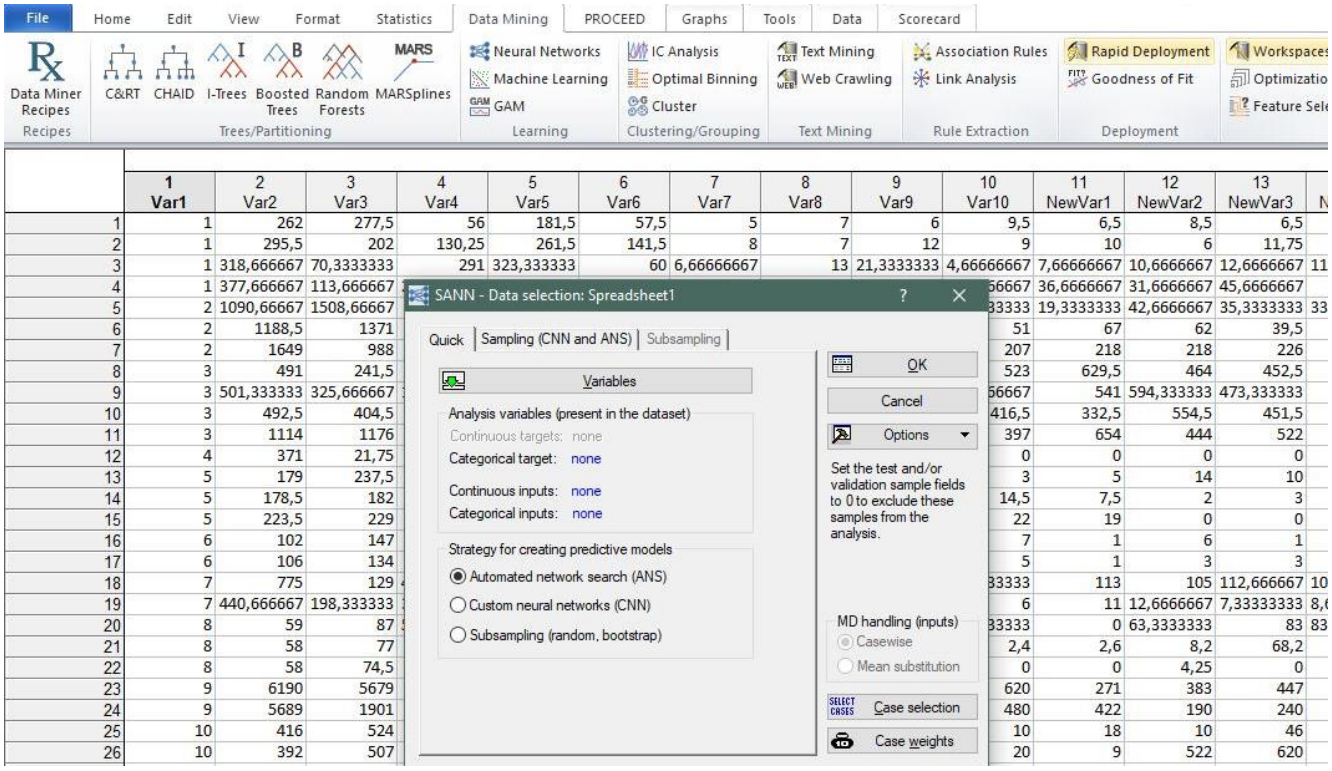
4.2.2 Точность идентификации при использовании метода сравнения сигнатур на основе машинного обучения

Подход к построению унифицированных ЭСП и СИИФ на основе распределения одной ассемблерной команды на равных интервалах и идентификации при помощи искусственной нейронной сети.

Эксперимент проводился для ассемблерной команды *jmp* и с использованием программного средства STATISTICA, которая позволяет настроить пользовательскую нейронную сеть и предоставляет удобный интерфейс взаимодействия с пользователем.

Обучение и тестирование нейронной сети предполагает определенное формирование входных данных. На рисунке 20 представлен скриншот окна программного средства STATISTICA, где в первом столбце приведены числовые

идентификаторы имен различных программ (их соответствие представлено в приложении 2), далее следуют унифицированные ЭСП встречаемости признака (одной ассемблерной команды на 30 равных интервалах разбиения), затем СИИФ, а в последнем столбце стоит отметка о принадлежности каждой из сигнатур к обучающей или тестовой выборке.



	1 Var1	2 Var2	3 Var3	4 Var4	5 Var5	6 Var6	7 Var7	8 Var8	9 Var9	10 Var10	11 NewVar1	12 NewVar2	13 NewVar3
1	1	262	277,5	56	181,5	57,5	5	7	6	9,5	6,5	8,5	6,5
2	1	295,5	202	130,25	261,5	141,5	8	7	12	9	10	6	11,75
3	1	318,666667	70,3333333	291	323,333333	60	6,66666667	13	21,3333333	4,66666667	7,66666667	10,6666667	12,6666667
4	1	377,666667	113,666667							6667	36,6666667	31,6666667	45,6666667
5	2	1090,66667	1508,66667							3333	19,3333333	42,6666667	35,3333333
6	2	1188,5	1371							51	67	62	39,5
7	2	1649	988							207	218	218	226
8	3	491	241,5							523	629,5	464	452,5
9	3	501,333333	325,666667							6667	541	594,333333	473,333333
10	3	492,5	404,5							416,5	332,5	554,5	451,5
11	3	1114	1176							397	654	444	522
12	4	371	21,75							0	0	0	0
13	5	179	237,5							3	5	14	10
14	5	178,5	182							14,5	7,5	2	3
15	5	223,5	229							22	19	0	0
16	6	102	147							7	1	6	1
17	6	106	134							5	1	3	3
18	7	775	129							3333	113	105	112,666667
19	7	440,666667	198,333333							6	11	12,6666667	7,33333333
20	8	59	87							3333	0	63,3333333	83
21	8	58	77							2,4	2,6	8,2	68,2
22	8	58	74,5							0	0	4,25	0
23	9	6190	5679							620	271	383	447
24	9	5689	1901							480	422	190	240
25	10	416	524							10	18	10	46
26	10	392	507							20	9	522	620

Рисунок 20 – Инициализация искусственной нейронной сети в программе STATISTICA

В пользовательской нейронной сети предлагается выбор типа архитектуры нейронной сети, число нейронов и итераций алгоритм обучения и другое.

По результатам ряда экспериментов было установлено, что наиболее точные показания идентификации достигаются при установке следующих параметров обучения искусственной нейронной сети (hbceyjr 21):

– тип сети: многослойный перцептрон (MLP);

– алгоритм обучения: метод сопряженных градиентов (Conjugate gradient);

– число итераций: 1000;

– число скрытых нейронов: 50.

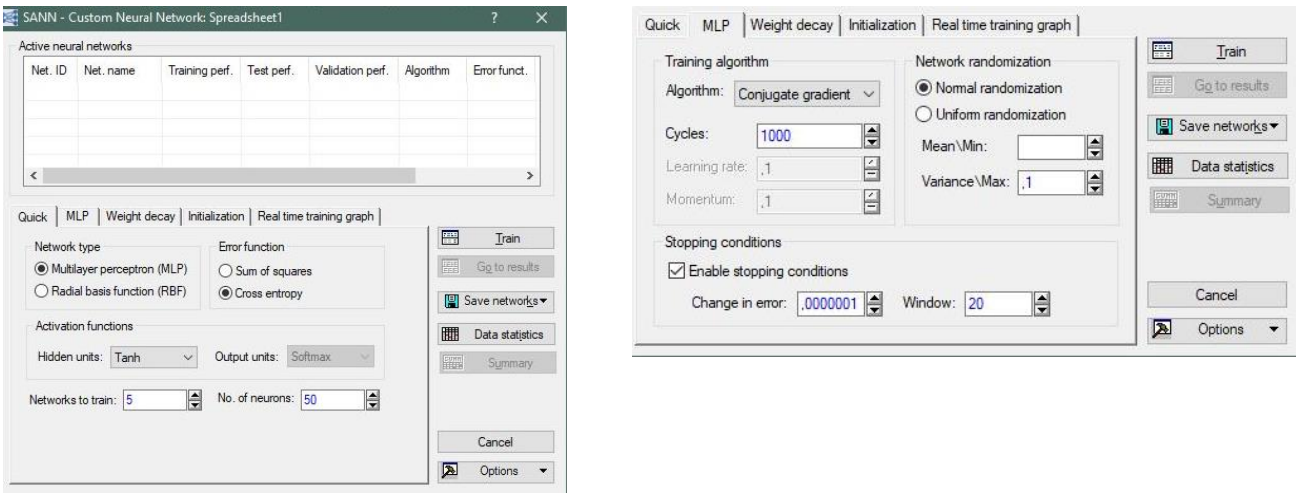


Рисунок 21 – Задание параметров нейронной сети

В ходе эксперимента было построено 5 моделей нейронных сетей (Модель 1-Модель 5). Результаты правильной классификации которых показаны в таблице11.

Таблица 11 – Показатели *Accuracy* для пяти моделей нейронной сети по ассемблерной команде *jmp*

	Accuracy
Модель 1	75,61
Модель 2	76,42
Модель 3	64,23
Модель 4	74,80
Модель 5	78,86
Среднее значение *Accuracy*	73,98

На рисунке 22 представлен график, отображающий распределение результатов произведенной классификации СИИФ из тестовой выборки. На оси абсцисс отмечены спрогнозированные классы (имена программ) для идентифицируемых файлов, на оси ординат – реальные классы (имена программ) этих файлов. Так, в случае совпадения прогнозируемого результата с реальным, на графике маркер ставится в точке с координатой (x, y), где $x = y$, т.е. на диагонали.

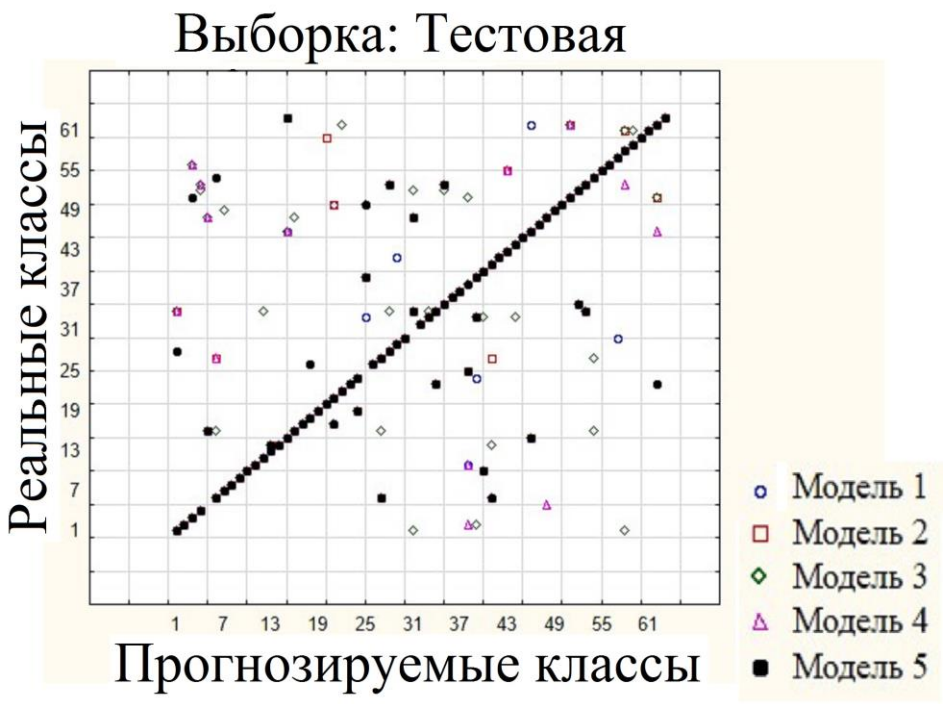


Рисунок 22 – График зависимости результатов идентификации от истинной принадлежности файлов классам

Общие результаты эксперимента оценивались при помощи точности *Accuracy* по десяти ассемблерным командам с применением искусственной нейронной сети представлены в таблице 12.

Таблица 12 – *Accuracy* для подхода на основе искусственной нейронной сети

Ассемблерная команда	Среднее значение *Accuracy* по пяти моделям
add	76,42
and	69,11
call	74,8
cmp	82,11
je	78,05
jmp	73,98
lea	56,1
mov	74,8
pop	69,1
push	53,66

Из таблицы 12 следует, что наибольший показатель *Accuracy* для идентификации сигнатур ELF-файлов, построенных на основе частот выбранной ассемблерной команды, составляет 82,11% для ассемблерной команды *cmp*.

Подход к построению ЭСП и СИИФ на основе распределения одной ассемблерной команды на равных интервалах и идентификации при помощи градиентного бустинга деревьев решений.

Для данного подхода были выбраны три библиотеки, реализующие алгоритм градиентного бустинга деревьев решений LightGBM, CatBoost и XGBoost. Формирование обучаемой модели для решения задачи мульти-классификации [89] происходило по нескольким параметрам, значения для которых представлены в таблице 13. Остальные значения параметров моделей были установлены по умолчанию.

Таблица 13 – Параметры обучения моделей

Параметры обучения	Описание параметров	XGBoost	LightGBM	CatBoost
booster/ boosting/ boosting_type	тип бустинга	gbtree	gbdt	plain
eta/ learning_rate	скорость обучения, используемая для уменьшения шага градиентного спуска	0,3	0,3	0,7
-/-/ l2_leaf_reg	коэффициент регуляризации L2, используемый для расчета значения листов	-	-	1
max_depth/ depth	глубина дерева	2	7	2
num_round/ num_iterations/ iterations	число итераций бустинга	1000	1000	1000
objective/ loss_function	метрика оценки (функция потерь), используемая в обучении модели	Multi:SoftMax	MultiClass	MultiClass

Общие результаты эксперимента оценивались при помощи точности *Accuracy* по десяти ассемблерным командам с применением градиентного бустинга деревьев решений представлены в таблице 14.

Таблица 14 – *Accuracy* для подхода на основе градиентного бустинга деревьев решений

Ассемблерная команда	*Accuracy* для LightGBM	*Accuracy* для CatBoost	*Accuracy* для XGBoost
add	82,93	85,37	92,68
and	79,67	86,18	92,68
call	81,30	84,55	94,31
cmp	78,05	85,37	84,55
je	86,99	89,43	91,06
jmp	84,55	83,74	95,93
lea	74,80	76,42	91,87
mov	74,80	85,37	94,31
pop	78,86	83,74	88,62
push	74,80	82,93	89,43

Очевидно, что для почти всех ассемблерных команд число верно идентифицированных программ выше при использовании библиотеки XGBoost и достигает максимального значения в 95,93% для команды *jmp* (118 правильно идентифицированных исполняемых файла из 123).

При этом из рисунка 23 видно значительное преимущество LightGBM во времени, затрачиваемого на обучение модели и идентификацию всех исполняемых файлов тестовой выборки.

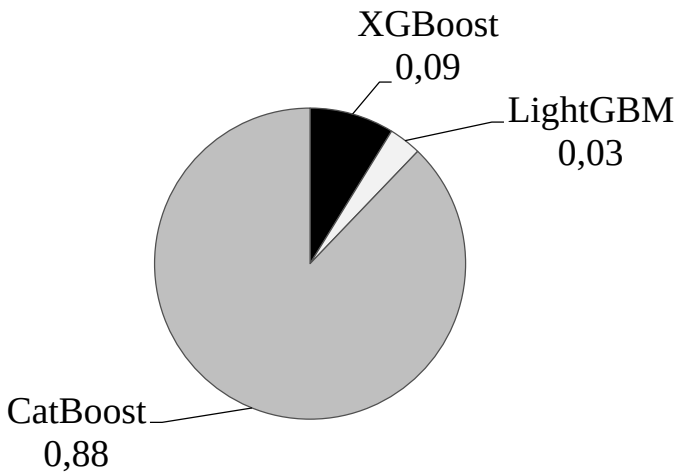


Рисунок 23 – Отношение времени обучения модели классификации и идентификации файлов тестовой выборки

4.2.3 Повышение точности идентификации путем использования аддитивного критерия Фишберна

В задачах, когда эффективность принимаемых решений оценивается не одним, а несколькими критериями, является центральной проблемой теории принятия решений. В работе [97] даются строгие определения равенства и неравенства критериев по важности. Классификация методов определения коэффициентов важности критериев представлена в работе [98], среди которых выделяется две группы методов: первые – определяют коэффициенты важности критериев, использование которых возможно в обобщенных свертках; и вторые – определяют весовые коэффициенты, использование которых в обобщенных свертках не рекомендуется.

Аддитивный критерий Фишберна [83], [100] как раз относится к первой группе методов, а в частности к методам аддитивной свертки. Данный критерий обладает рядом достоинств, в частности, отсутствует необходимость в опросе мнений экспертов, отсутствуют ограничения на условия реализации, простота расчетов и т.д.

Методы аддитивной свертки можно использовать, если функция полезности $\varphi(f(x))$ представима в аддитивной форме:

$$\varphi\left(f_1(x), \dots, f_n(x)\right) = \sum_{i=1}^{n} \lambda_i \cdot \varphi_i \cdot (f_i(x))$$

где λ – коэффициенты относительной важности критериев, которые определяются по формуле:

$$\lambda_i = \frac{2 \cdot (n - i + 1)}{n \cdot (n + 1)}, i = 1, \dots, n$$

при упорядоченных критериях $f_1(x) \geq \dots \geq f_n(x)$.

При этом, если значения нескольких критериев равны $f_i(x) = f_{i+1}(x) = \dots = f_{i+m}(x)$, тогда и коэффициенты относительной важности для них одинаковы и равны среднему значению $\frac{1}{\sum j} \cdot \sum_{j=1}^{i+m} \lambda_j$, как если бы они были

рассчитаны для не равных критериев с сохранением их порядка $f_i(x) > f_{i+1}(x) > \ldots > f_{i+m}(x)$.

По результатам проведенных экспериментов можно сделать вывод о том, что наиболее точные результаты идентификации были получены при построении сигнатур, основанных на частоте одной ассемблерной команде на равных интервалах разбиения, с использованием метода их сравнения на основе градиентного бустинга деревьев решений в реализации библиотеки XGBoost. В таблице 13 представлены результаты по десяти ассемблерным командам выделенные цветовой шкалой, так набольшая точность идентификации исполняемых файлов достигается при использовании ассемблерной команды *jmp*, а наихудшая при – *cmp*.

Таблица 15 – Упорядоченные цветом ассемблерные команды по достигаемой точности идентификации

xgboost									
add	and	call	cmp	je	jmp	lea	mov	pop	push
113	113	115	104	112	118	112	116	109	110

Произведем расчет коэффициентов относительной важности критериев (ассемблерных команд) для нашей задачи.

jmp *mov* *call* *add* *and*

$$\lambda_1=\frac{2\cdot(10-1+1)}{10\cdot(10+1)} \quad \lambda_2=\frac{2\cdot(10-2+1)}{10\cdot(10+1)} \quad \lambda_3=\frac{2\cdot(10-3+1)}{10\cdot(10+1)} \quad \lambda_4=\frac{2\cdot(10-4+1)}{10\cdot(10+1)} \quad \lambda_5=\frac{2\cdot(10-5+1)}{10\cdot(10+1)}$$

λ_1=0,182 λ_2=0,164 λ_3=0,145 λ_4=0,118 λ_5=0,118

je *lea* *push* *pop* *cmp*

$$\lambda_6=\frac{2\cdot(10-6+1)}{10\cdot(10+1)} \quad \lambda_7=\frac{2\cdot(10-7+1)}{10\cdot(10+1)} \quad \lambda_8=\frac{2\cdot(10-8+1)}{10\cdot(10+1)} \quad \lambda_9=\frac{2\cdot(10-9+1)}{10\cdot(10+1)} \quad \lambda_{10}=\frac{2\cdot(10-10+1)}{10\cdot(10+1)}$$

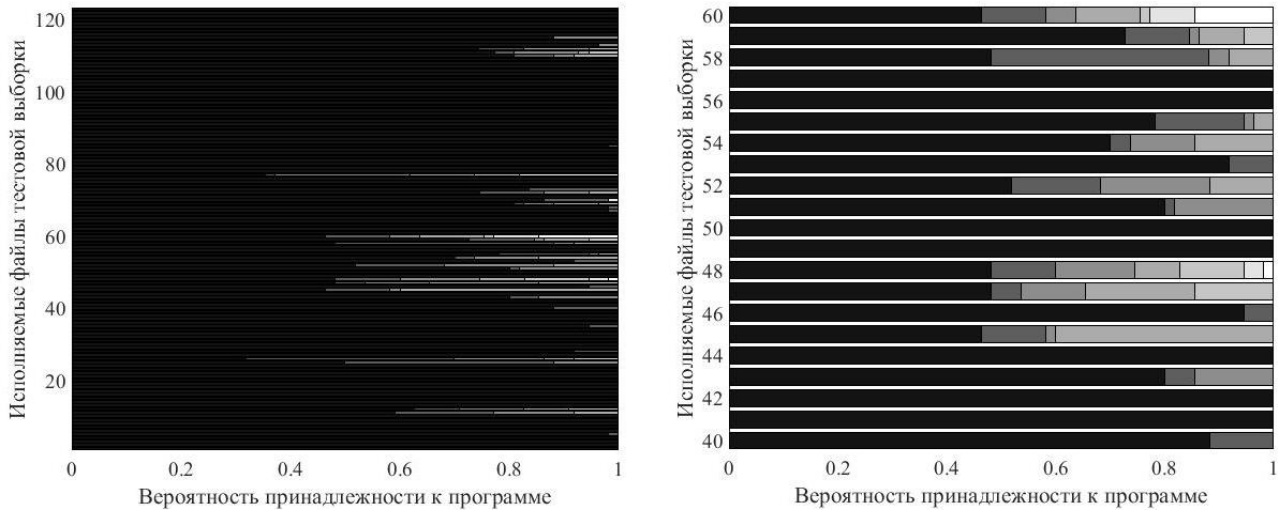
λ_6=0,082 λ_7=0,082 λ_8=0,055 λ_9=0,036 λ_{10}=0,018

Таким образом, мы получаем аддитивную функцию вида:

$$\varphi(jmp,\ldots,cmp)=0{,}182\cdot jmp+0{,}164\cdot mov+0{,}145\cdot call+0{,}118\cdot add+0{,}118\cdot and+$$

$$+0{,}082\cdot je+0{,}082\cdot lea+0{,}055\cdot push+0{,}036\cdot pop+0{,}018\cdot cmp$$

применив которую к результатам классификации XGBoost получим вероятности принадлежности исполняемых файлов тестовой выборки к определенным программам на основании совокупности десяти ассемблерных команд и коэффициентов их важности. Так на рисунке 24 отображены результаты идентификации с постобработкой результатов классификации XGBoost по всем 123 исполняемым файлам тестовой выборки, отмеченным на оси ординат, по оси абсцисс отмечается вероятность принадлежности исполняемого файла к классу (программе).



а) б)

Рисунок 24 – Результаты постобработки результатов классификации
а) для всех файлов; б) для 20 файлов для ближайшего рассмотрения

Автором, для более понятной визуализации, специально была выбрана линейчатая диаграмма с накоплением, где первым прямоугольником отображается вероятность принадлежности исследуемого файла к истинной программе (которой он и является). Далее следуют вероятности принадлежности к другим, не истинным, программам. Так, после применения аддитивного критерия Фишберна,

точность идентификации исполняемых файлов возросла до 99,19% (только один файл из 123 был неправильно идентифицирован) [76].

4.3 Определение итоговой точности идентификации программного обеспечения на основе предложенной методики

Цель эксперимента – определить точность идентификации исполняемых файлов по характеристикам их ассемблерного кода с использованием разработанной методики в условиях ограниченного количества версий и программ разнообразной функциональности, а также провести сравнение с результатами, других исследовательских работ.

На унифицированных сигнатурах программ наилучшим методом идентификации, при подходе на основе оценки матрицы ошибок (*Confusion Matrix*) и уровне значимости $p = 0,05$, показал себя критерий однородности хи-квадрат с построением сигнатур на основе 118 ассемблерных команд, чья точность достигла 98,68%, что выше на 0,12% и на 3,59%, чем при использовании критерия однородности хи-квадрат и согласия Колмогорова, соответственно, с построением сигнатур на основе частоты встречаемости одной ассемблерной команды.

Однако, при расчете точности при помощи *Accuracy*, достигаемые результаты равны лишь 79,55%, 78,05% и 50,41% соответственно. Точность же с использованием искусственной нейронной сети достигает 82,11% и является наиболее эффективной с точки зрения числа верно идентифицированных исполняемых файлов, при построении унифицированных сигнатур.

На не унифицированных сигнатурах методы градиентного бустинга деревьев решений, при подходе к оценке через *Accuracy*, показали наиболее высокую достигаемую точность со средним показателем равным 84,98%, чем с использованием статистических критериев однородности хи-квадрат и Колмогорова, а также с использованием нейронной сети, чей средний показатель равен 66,07%. Среди подходов к реализации градиентного бустинга, наиболее результативным является XGBoost со средним показателем точности 91,29%.

Среди ассемблерных команд наилучший результат получается при использовании в качестве признака команды *je* с 82,93%, наихудший при использовании *push* с 71,54%. Так разница между достигаемыми показателями точности для этих команд составляет 11,39%.

Максимально достигаемая величина значения точности идентификации, через оценку *Accuracy*, равняется 95,95% и достигается за счет применения метода идентификации на основе градиентного бустинга деревьев решений в реализации XGBoost к сигнатурам, сформированным на основе распределения одной ассемблерной команды *jmp* на равных интервалах дизассемблированного кода программы.

Таблица 16 – Показатель *Accuracy* по каждой ассемблерной команде, в процентах и *F-measure*, (в скобках)

Ассемблерные команды	Статистические критерии, $p = 0,05$			Машинное обучение			
	однородности хи-квадрат	согласия Колмогорова	однородности хи-квадрат	Нейронная сеть MLP	Градиентный бустинг деревьев решений		
					LightGBM	CatBoost	XGBoost
add	79,55	-	40,65	76,42	82,92	85,37	92,68
and		-	60,16	69,11	78,87	86,18	91,87
call		-	65,85	74,80	81,3	84,55	93,50
cmp		50,41	70,73	*82,11*	78,04	85,37	84,55
je		-	69,11	78,05	*86,99* (0,84)	*89,43* (0,88)	91,05
jmp		-	65,85	73,98	83,74	83,74	*95,93* (0,96)
lea		-	*78,05*	56,10	73,99	76,42	91,06
mov		-	47,97	74,80	73,99	85,37	94,30
pop		-	60,16	69,11	78,86	83,74	88,60
push		-	56,91	53,66	74,79	82,93	89,40
Фишберн	-	-	-	84,93	87,81	91,87	*99,19* (0,99)

После применения подхода постобработки результатов, на основе аддитивного критерия Фишберна и сочетания десяти ассемблерных команд, точность идентификации исполняемых файлов решений в реализации XGBoost возрастает на 3,26% и достигает показателя 99,19%.

Оценка точности по различным ассемблерным командам приведена в таблице 16. *Accuracy* рассчитана для каждой ассемблерной команды, *F-measure* только для наилучших результатов.

Оценка точности идентификации на основе *Accuracy* и *F-measure*, позволяет сделать вывод о том, что разработанная методика идентификации ELF-файлов позволяет получить белее высокий показатель точности с использованием всех рассмотренных методов классификации в сравнении с существующими методами идентификации программ, описанными в рассмотренных работах отечественных и зарубежных авторов. Так разработанная автором методика, не только позволяет решить совершенно другую область распознавания ПО, но и превосходит другие наиболее результативные подходы (таблица 17).

Таблица 17 – Точность идентификации ПО с использованием разработанной методики и различных современных методов

Признаковое пространство и метод идентификации	Число классов	Оценка качества метода	
		Точность (Accuracy), %	**Бикубическая мера качества кластеризации**
Печатаемые строки + Naive Bayes, [35]	Бинарная классификация	97,11	-
Последовательность частоты встречаемости очередности (n = 2) ассемблерных команд + SVM: normalised polynomial, [42]	Бинарная классификация	95,9	-
Вектора для блоков постоянного размера побайтового кода программы + Редакционное расстояние, [30]	Мульти-классификация	-	0,69
Последовательность частоты встречаемости одной ассемблерной команды на равных интервалах + XGBoost, [75]	Мульти-классификация	95,93	0,96
Последовательность частоты встречаемости одной ассемблерной команды на равных интервалах + XGBoost + Фишберн, [76]	Мульти-классификация	99,19	0,99

4.4 Использование результатов исследования для повышения информационной безопасности автоматизированных систем

Методика идентификации ПО может быть применена для повышения безопасности информационных систем организации, путем внедрения ее в качестве технической меры по аудиту ПО на электронных носителях информации.

Контроль установленных программ представляет собой важную составляющую в организации защищенного функционирования бизнес-процессов. Методика позволяет осуществлять автоматизированную идентификацию исполняемых файлов формата ELF в соответствии с формируемых архивом легитимных или не легитимных программ.

До применения методики реализация угрозы со стороны злоумышленника может быть произведена, как показано на рисунке 25 [101].

Рисунок 25 – Схема возможного снижения уровня безопасности системы и воздействие злоумышленника на ИС организации

Группа пользователей АС взаимодействует с СВТ и другими электронными носителями информации, присутствующими в организации и соединенными с

главным сервером (или серверами, хранящими и обрабатывающими защищаемую информацию). В условиях соблюдения установленной политики безопасности, службе защиты информации известны присущие ИС уязвимости и сформирована стратегия по минимизации возможных рисков при инцидентах ИБ. Однако, при появлении несанкционированно установленного ПО, присутствует вероятность возникновения новой уязвимости в системе, существование которой не будет известно службе безопасности до тех пор, пока не будет проведен повторный мониторинг АС для выявления уязвимостей.

В этот промежуток времени злоумышленник, обладая необходимыми знаниями и средствами, способен произвести атаку на сервер организации путем эксплуатации возникшей уязвимости из-за несанкционированной установки ПО.

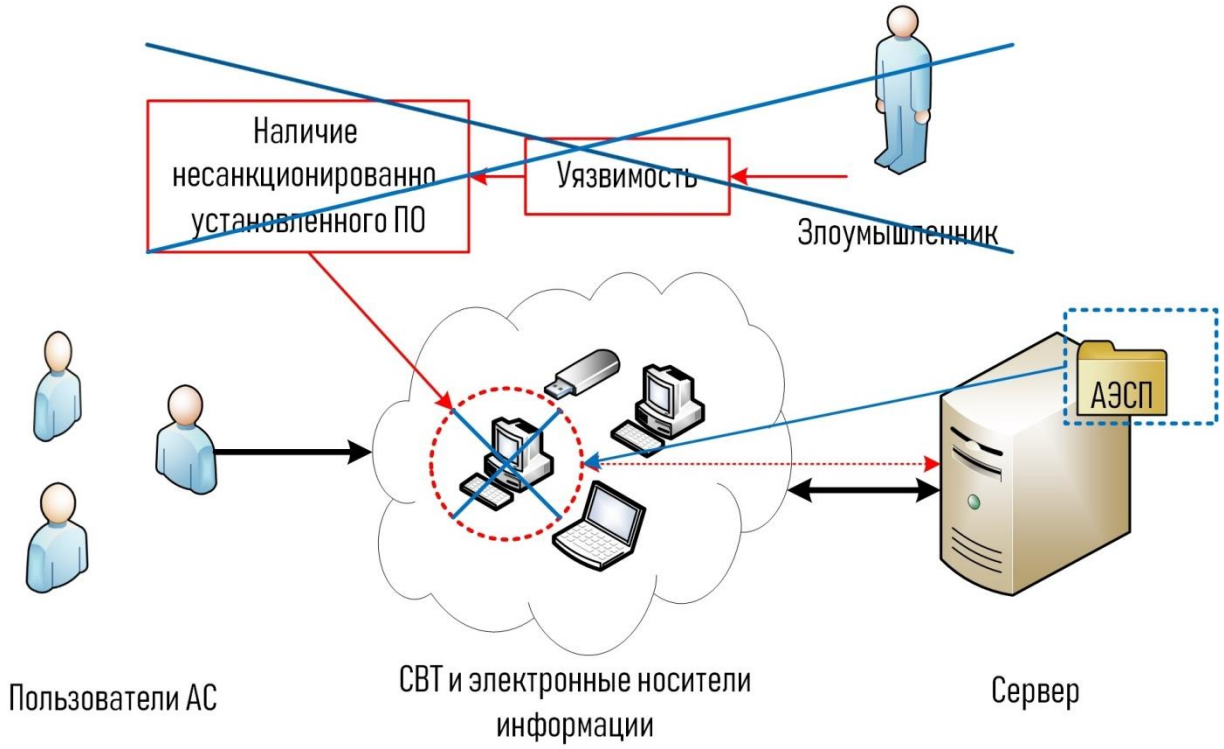


Рисунок 26 – Схема возможного снижения уровня безопасности системы и воздействие злоумышленника на ИС организации, включающая дополнительную меру по аудиту ПО

Применение разработанной методики в ИС организации способна предотвратить описанный сценарий путем проведения автоматизированного

аудита установленного ПО на СВТ и электронных носителях информации (рисунок 26).

Так располагая АЭСП на стороне сервера и проводя сравнения СИИФ всех присутствующих программ с каждого СВТ и подключаемых к ним съемных электронных носителей информации можно в короткие сроки уведомить службу ИБ о нарушении правил безопасности и принять своевременные защитные меры.

Разработанные методы и методика могут быть применены для выявления факта нарушения установленных организационных мер о запрете на установку определенных программ путем распознавания установленного исполняемого файла как ту или иную программу в АЭСП или не входящую в данный архив.

Можно выделить смежные задачи, решаемые разработанной методикой:

– выявление не добросовестных пользователей, многократно устанавливающих запрещенное ПО;

– выявление пользователей, пытающихся обойти меры по обеспечению ИБ;

– анализ конкретной программы, на основе версий других, схожих по функционалу программ, для выявления степени принадлежности их области назначения.

Стоит отметить, что методика эффективно показывает себя на ограниченном наборе данных и способна идентифицировать исполняемый файл, не задействованный при создании АЭСП, имеющий новую версию или внесенные в него изменения, а также имеющий исправленные, или не имеющий их вообще, метаданные.

Методика может быть реализована в программном комплексе и использоваться как при периодически проводимых мероприятиях, так и быть установлена на компьютере пользователя для проведения идентификации всего устанавливаемого ПО в автоматическом режиме.

Методика позволяет производить идентификацию непосредственно исполняемого файла, а не его метаданных, записанных в конфигурационных файлах, и, таким образом, может быть использована в компьютерной криминалистике специальными службами.

Выводы по главе 4

а) Для выполнения цели экспериментов при подготовки настоящего диссертационного исследования был произведен сбор и анализ экспериментальных данных, включающий разнообразные виды ПО Linux ОС и их различных версий.

б) Анализ разработанных методов идентификации ПО позволил выявить наилучший из них, основанный на формировании сигнатур программ с подсчетом частоты встречаемости выбранных десяти ассемблерных команд на равных интервалах разбиения дизассемблированного кода исполняемых файлов и использующий алгоритм градиентного бустинга деревьев решений с последующей постобработкой получаемых результатов классификации.

в) Сравнение точности производимой идентификации исполняемых файлов при использовании разработанного метода по формированию ЭСП и СИИФ с наилучшими результатами, получаемыми другими исследователями, позволяет сделать вывод о наличии более высокой точности идентификации исполняемых файлов перед методами, основанными на бинарной классификации на 2,08% измеренными в *Accuracy*, и перед методами, основанными на мульти-классификации на 18% измеренными в *F-measure*.

г) Подход на основе машинного обучения показал наиболее низкую вычислительную сложность, перед подходом с использованием статистических критериев, что подтверждается проведенными экспериментами.

д) Анализ результатов применения градиентного бустинга деревьев решений на сформированных ЭСП и СИИФ позволяет сделать вывод о том, что данный класс алгоритмов машинного обучения способен достигать наиболее точных результатов идентификации ПО. Применение методики идентификации с данным алгоритмом позволяет в среднем достигать на 4,10% более точные результаты, рассчитанные при помощи *Accuracy*, и на 11,42% более точные результаты, рассчитанные при помощи *Accuracy* при использовании библиотеки XGBoost.

е) Постобработка результатов классификации для алгоритмов машинного обучения показывает, что использование аддитивного критерия Фишберна с расчетом коэффициентов важности для десяти ассемблерных команд, позволяет в среднем повысить результаты классификации на 4,17%, а для библиотеки XGBoost достигать почти идеальной идентификации с *Accuracy* равным 99,19%.

Заключение

В диссертационной работе решена научная задача увеличения точности идентификации исполняемых файлов, устанавливаемых на средства вычислительной техники, в условиях наличия различных версий ПО, большого числа различных наименований программ и ограниченности числа объектов обучающей выборки, обусловленных реальным состоянием выпускаемых версий того или иного ПО. Использование разработанной методики позволяет увеличить точность идентификации исполняемых файлов, что предоставляет возможность производить более тщательный анализ электронных носителей информации на предмет наличия на них нелегитимно установленного ПО. Методика реализуется как часть технических мер по обеспечению ИБ и предотвращению потенциального возникновения новых уязвимостей ИС, способных повлечь за собой негативные воздействия на защищаемую информацию и персонал АС. В том числе получены следующие *научные результаты*, составляющие *итоги* исследования:

1) Произведено исследование и анализ существующих подходов и методов идентификации программного обеспечения, используемых отечественными и зарубежными исследователями, выявлены достоинства и недостатки данных методов, а также проанализированы условия и ограничения их использования.

2) Рассмотрены ELF формат файлов Linux систем, представление программы в виде ассемблерного кода и распределение ассемблерных команд. Проанализированы характеристики ассемблерных команд, их частота и информативность, а также потенциал их применения при формировании сигнатур исполняемых файлов и их последующего сравнения.

3) Разработан метод представления идентификатора исполняемого файла в виде сигнатуры, позволяющий реализовывать гибкий подход к идентификации версий ПО.

4) Разработан метод сравнения сигнатур программ на основе использования библиотеки градиентного бустинга деревьев решений с последующей постобработкой результатов классификации, на основе аддитивного критерия.

5) Разработана методика идентификации исполняемых файлов, позволяющая повысить точность идентификации в условиях большого числа классов и малого числа элементов обучающей выборки, соответствующих реальным условиям эксплуатации ПО в организации. Достигается значение *Accuracy* в 99,19%, что на 2,08% превышает бинарный подход классификации на основе использования UNIX команды strings, показывающая печатаемые строки в объекте или двоичном файле, и сравнения получаемых сигнатур при помощи наивного Байесовского классификатора. Достигаемые результаты также превышают измеренный показатель *F-measure* на 30% чем у мульти-классификационного подхода на основе использования вектора для блоков постоянного размера побайтового кода программы и сравнения получаемых сигнатур при помощи редакционного расстояния.

Рекомендации по применению результатов работы для идентификации исполняемых файлов в автоматизированных системах включают в себя указания по формированию архива сигнатур; применению метода и методики идентификации исполняемых файлов, позволяющих обнаруживать нарушения установленных мер политики безопасности в плане запрета на несанкционированную установку программного обеспечения. Также рекомендации включают разработку подходов, направленных на повышение эффективности процесса аудита электронных носителей информации и его автоматизации, позволяющие идентифицировать исполняемые файлы, не основываясь на методах оценки их целостности и анализе метаданных встроенными средствами операционных систем.

В качестве *перспектив дальнейшей разработки тематики* можно выделить исследования, связанные с разработкой динамического подхода к идентификации программного обеспечения в близких к системе реального времени ограничениях на скорость реагирования при загрузке ELF-файлов на компьютер пользователя, до момента окончательного скачивания полной версии файла. Такой подход позволит производить превентивную меру по защите автоматизированной системы от несанкционированной установки запрещенного программного обеспечения.

Положения, выносимые на защиту, соотнесены с пунктами паспорта специальности 05.13.19 — «Методы и системы защиты информации, информационная безопасность»: «6. Модели и методы формирования комплексов средств противодействия угрозам хищения (разрушения, модификации) информации и нарушения информационной безопасности для различного вида объектов защиты вне зависимости от области их функционирования» (результаты 4-5); «13. Принципы и решения (технические, математические, организационные и др.) по созданию новых и совершенствованию существующих средств защиты информации и обеспечения информационной безопасности» (результаты 2-3); «14. Модели, методы и средства обеспечения внутреннего аудита и мониторинга состояния объекта, находящегося под воздействием угроз нарушения его информационной безопасности» (результат 5).

Список сокращений и условных обозначений

API – программный интерфейс приложения (application programming interface)

AUC – площадь под roc-кривой (area under roc curve)

CRC – циклический избыточный код (cyclic redundancy check)

CVM – метод опорных векторов (support vector machine)

DLL – динамически подключаемая библиотека (dynamic link library)

ELF – формат исполнимых и компонуемых файлов (Executable and Linkable Format)

HAM – управление аппаратными активами (hardware asset management)

IoT – интернет вещей (internet of things)

ITAM – управление активами информационных технологий (information technology asset management)

ITAM – управление ИТ-активами (IT asset management)

ROC – рабочая характеристика приёмника (receiver operating characteristic)

SAM – управление активами программного обеспечения (software asset management)

SAM – управление программными активами (software asset management)

АС – автоматизированная система

АЭСП – архив эталонных сигнатур программ

ЗИ – защита информации

ИБ – информационная безопасность

ИС – информационная система

ОС – операционная система

ПО – программное обеспечение

СИИФ – сигнатуры идентифицируемого исполняемого файла

СП – сигнатура программы

УК РФ – уголовный кодекс российской федерации

УСП – унифицированных сигнатур программ

ЭВМ – электронно-вычислительная машина

ЭСП – эталонной сигнатурой программы

Список использованной литературы

1. Tool Interface Standards (TIS). Executable and Linking Format (ELF) Specification Version 1.2. TIS Commit. 1995. 106 с.
2. ГОСТ Р ИСО/МЭК 27002-2012 Информационная технология (ИТ). Методы и средства обеспечения безопасности. Свод норм и правил менеджмента информационной безопасности. М.: Стандартинформ, 2012.
3. ГОСТ Р ИСО/МЭК 27001-2006 Информационная технология (ИТ). Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Требования. М.: Стандартинформ, 2006.
4. ГОСТ Р 50922-2006. Защита информации. Основные термины и определения. М.: Стандартинформ, 2006.
5. ФЗ от 27 июля 2006 г. № 149-ФЗ «Об информации, информационных технологиях и о защите информации (с изменениями на 18 декабря 2018 года)». Собрание законодательства Российской Федерации, N 31 (ч.I), 31.07.2006, ст.3448, 2006.
6. Закон РФ от 21 июля 1993 г. №5485-1 «О государственной тайне (с изменениями на 29 июля 2018 года)». Собрание законодательства Российской Федерации, N 41, 13.10.1997, ст.4673, 1993.
7. Указ Президента РФ от 6 марта 1997 г. №188 «Об утверждении перечня сведений конфиденциального характера (с изменениями на 13 июля 2015 года)». Собрание законодательства Российской Федерации, N 10, 10.03.97, ст.1127, 1997.
8. ФЗ от 20 февраля 1995 г. № 24-ФЗ «Об информации, информатизации и защите информации (с изменениями на 10 января 2003 года) (утратил силу с 09.08.2006 на основании Федерального закона от 27.07.2006 N 149-ФЗ)». Собрание законодательства Российской Федерации N 8, 20.02.95, ст.609. ст.2 с.
9. Петренко С.А., Курбатов В.А. Политики безопасности компании при работе в Интернет. 2011. 396 с.

10. Boukhtouta A. и др. Graph-theoretic characterization of cyber-threat infrastructures // Digit. Investig. 2015. Т. 14, № 1. С. 3–15.
11. Бондарев В. Введение в информационную безопасность автоматизированных систем. 2018. 252 с.
12. Software Management: Security Imperative, Business Opportunity [Электронный ресурс]. 2018. С. 24. URL: https://gss.bsa.org/wp-content/uploads/2018/05/2018_BSA_GSS_Report_en.pdf. Режим доступа: свободный, дата обращения: 18.03.2018.
13. Отчет о видах наказания по наиболее тяжкому преступлению (без учета сложения) [Электронный ресурс]. 2017. URL: http://www.cdep.ru/index.php?id=79&item=4572. Режим доступа: свободный, дата обращения: 11.12.2018.
14. Базовая модель угроз безопасности персональных данных при их обработке в информационных системах персональных данных (выписка). ФСТЭК России, 2008 год. 2008.
15. Быкова В.В., Катаева А.В. Методы и средства анализа информативности признаков при обработке медицинских данных. // Журнал Программные продукты и системы / Softw. Syst. 2016. Т. 2, № 114. С. 172–178.
16. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: Data Mining, Inference, and Prediction. // Springer. 2017. 745 с.
17. Еремеев М.А., Кравчук А.В. Анализ методов распознавания вредоносных программ. // Журнал Вопросы защиты информации. 2014. Т. 3. С. 44–51.
18. Giesen D. Philosophy and practical implementation of static analyzer tools // Softw. Dev. J. 1998. № 7. С. 12–31.
19. Christopher C.N. Evaluating Static Analysis Frameworks. 2003. 213 с.
20. Seo K. и др. Detecting Similar Files Based on Hash and Statistical Analysis for Digital Forensic Investigation // Digital Forensic Investigation. In: Computer Science and its Applications (CSA 2009). 2009. С. 1–6.
21. Breitinger F. B.H. Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2 // Digital Forensics and Cyber Crime. ICDF2C 2012. Lecture

Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. 2012. C. vol 114.

22. Kornblum J.D. Identifying almost identical files using context triggered piecewise hashing // Digit. Investig. 2006. Т. 3. С. 91–97.
23. Long C., Guoyin W. An Efficient Piecewise Hashing Method for Computer Forensics // First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008). 2008. С. 635–638.
24. Baier H., Frank B. Security Aspects of Piecewise Hashing in Computer Forensics // 2011 Sixth International Conference on IT Security Incident Management and IT Forensics. С. 21–36.
25. Цифровые подписи в Windows 7 [Электронный ресурс]. 2012. URL: http://windowsnotes.ru/windows-7/cifrovye-podpisi-v-windows-7. Режим доступа: свободный, дата обращения: 23.04.2018.
26. Niemela J. It’s Signed, therefore it’s Clean, right? [Электронный ресурс] // Protecting the irreplaceable, f-secure.com. 2010. С. 1–30. URL: https://www.f-secure.com/weblog/archives/Jarno_Niemela_its_signed.pdf. Режим доступа: свободный, дата обращения: 23.04.2018.
27. Sorokin I. Comparing files using structural entropy // J. Comput. Virol. Hacking Tech. 2011. Т. 7. С. 259–265.
28. Копыльцов А.В., Сорокин И.В. Вейвлет-анализ структурной энтропии файлов // Известия Российского государственного педагогического университета им. А.И. Герцена. 2011. Т. 138. С. 7–15.
29. Копыльцов А.В., Сорокин И.В. Алгоритм выравнивания последовательностей сегментов файлов // Вестник ИНЖЭКОНа. Серия: Технические науки. 2011. Т. 8. С. 31–36.
30. Антонов А.Е., Федулов А.С. Идентификация типа файла на основе структурного анализа // ПРИКЛАДНАЯ ИНФОРМАТИКА. 2013. Т. 2, № 44. С. 68–77.
31. Сорокин И.В., Копыльцов А.В. Модели распознавания вредоносных программ на основе энтропийных характеристик // Региональная

информатика и информационная безопасность. 2015. С. 313–316.

32. Сорокин И.В., Копыльцов А.В. Использование неокогнитрона для распознавания вредоносных файлов // Известия СПБГЭТУ ЛЭТИ. 2013. Т. 3. С. 45–51.
33. Казарин О.В. Безопасность программного обеспечения компьютерных систем. М.: МГУЛ, 2003. 212 с.
34. Гроувер Д. и др. Защита программного обеспечения: Пер. с англ / под ред. Д.Гроувера. 1992. 288 с.
35. Schultz M.G. и др. Data mining methods for detection of new malicious executables // Proceedings of the IEEE Symposium on Security and Privacy. Los Alamitos, CA, 2001. С. 38–49.
36. Cohen W.W. Fast effective rule induction // Proceedings of the Twelfth International Conference on Machine Learning. San Francisco, CA, 1995. С. 115–123.
37. Kolter J.., Maloof M.. Learning to detect and classify malicious executables in the wild // J. Mach. Learn. Res. 2006. Т. 7. С. 2721–2744.
38. Bergeron J. и др. Static analysis of binary code to isolate malicious behavior // 1999 Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises. 1999. С. 184–189.
39. Christodorescu, Mihai Jha S. Static Analysis of Executables to Detect Malicious Patterns // 12th USENIX Security Symposium. 2003. С. 169–186.
40. Christodorescu M. и др. Semantics-aware malware detection // 2005 IEEE Symposium on Security and Privacy. 2005. С. 32–46.
41. Bilar D. Opcodes as predictor for malware // Int. J. Electron. Secur. Digit. Forensics. 2007. Т. 1. С. 156–168.
42. Santos I. и др. Opcode sequences as representation of executables for data-mining-based unknown malware detection // Inf. Sci. (Ny). 2013. Т. 231. С. 64–82.
43. Shabtai A. и др. Detecting unknown malicious code by applying classification techniques on opcode patterns // Secur. Inform. 2012. Т. 1, № 1. С. 1–22.
44. Santos I. и др. OPEM: A Static-Dynamic Approach for Machine-Learning-Based

Malware Detection // International Joint Conference CISIS’12-ICEUTE´12-SOCO´12 Special Sessions. Berlin, Heidelberg, 2013. С. 271–280.

45. Darabian H. и др. An opcode-based technique for polymorphic Internet of Things malware detection // Concurr. Comput. Pract. Exp. 2019. С. 1–14.
46. Zhang H. и др. Classification of ransomware families with machine learning based on N-gram of opcodes // Futur. Gener. Comput. Syst. 2019. Т. 90. С. 211–221.
47. Li P. и др. On Challenges in Evaluating Malware Clustering // International Workshop on Recent Advances in Intrusion Detection. 2010. С. 238–255.
48. Ying-xu. L., Zeng-hui. L. Unknown Malicious Identification // Adv. Electr. Eng. Comput. Sci. Lect. Notes Electr. Eng. 2009. Т. 39. С. 301–312.
49. Ebringer T., Sun L., Boztas S. A Fast Randomness Test that Preserves Local Detail // 18th Virus Bulletin International Conference. United Kingdom, 2008. С. 34–42.
50. Willems C., Holz T., Freiling F. Toward automated dynamic malware analysis using cwsandbox // 2007 IEEE Symposium on Security and Privacy (S&P 2007). 2007. С. 32–39.
51. Bayer U., Kruegel C., Kirda E. Ttanalyze: A tool for analyzing malware // 15th European Institute for Computer Antivirus Research (EICAR 2006). 2006. С. 180–192.
52. Song D. и др. Bitblaze: A new approach to computer security via binary analysis // 4th International Conference on Information Systems Security. 2008. С. 1–25.
53. AVG [Электронный ресурс]. URL: https://www.avg.com/en-ww/avg-and-norman-business-products?_ga=2.1955633.1678056726.1552380509-211382991.1552380509.
54. Corporation S. Advanced Threat Protection [Электронный ресурс]. URL: https://www.symantec.com/products/threat-protection.
55. Lin C.H., Pao H.K., Liao J.W. Efficient dynamic malware analysis using virtual time control mechanics // Comput. Secur. 2018. Т. 73. С. 359–373.
56. Egele M. и др. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools // ACM Comput. Surv. 2012. Т. 44, № 2. С. 42.
57. Gray-Donald T.A., Price M.W. Date and time simulation for time-sensitive

applications: пат. US 8,418,151 B2 USA. USA, 2013. С. 9.

58. Bulazel A., Bülent Y. A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: Pc, mobile, and web. // the 1st Reversing and Offensive-oriented Trends Symposium. 2017. С. 1–21.
59. Windows Privilege Escalation Fundamentals [Электронный ресурс]. 2014. URL: http://www.fuzzysecurity.com/tutorials/16.html.
60. Вартанов С.П., Герасимов А.Ю. Динамический анализ программ с целью поиска ошибок и уязвимостей при помощи целенаправленной генерации входных данных // Труды ИСП РАН. 2014. Т. 26, № 1. С. 375–394.
61. Мельников П.В., Горюнов М.Н., Анисимов Д.В. Подход к проведению динамического анализа исходных текстов программ // Вопросы кибербезопасности. 2016. Т. 3, № 16. С. 33–39.
62. Klosterboer L. Implementing ITIL configuration management. 2007. 264 с.
63. England R. Owning ITIL. 2009. 178 с.
64. Bird J. DevOpsSec. 2016.
65. Samanage. Samanage [Электронный ресурс]. 2019. URL: https://www.samanage.com (дата обращения: 28.03.2019).
66. Kaspersky. Kaspersky Systems Management [Электронный ресурс]. 2019. URL: https://www.kaspersky.ru (дата обращения: 28.03.2019).
67. Microsoft. Microsoft Assessment and Planning Toolkit [Электронный ресурс]. 2018. URL: https://www.microsoft.com/en-us/Download/details.aspx?id=7826 (дата обращения: 28.03.2019).
68. FinalWire. AIDA64 [Электронный ресурс]. URL: https://www.aida64.com (дата обращения: 28.03.2019).
69. Lansweeper. Lansweeper [Электронный ресурс]. 2019. URL: https://www.lansweeper.com (дата обращения: 28.03.2019).
70. Bayer U. и др. Scalable, behavior-based malware clustering // Proceedings of the Network and Distributed System Security Symposium. 2009. С. 8–11.
71. Krivtsova I.E., Lebedev I.S., Salakhutdinova K.I. Identification of Executable Files on the basis of Statistical Criteria // Proceedings of the 20th Conference of Open

Innovations Association FRUCT, IET. 2017. С. 202–208.

72. Салахутдинова К.И., Лебедев И.С., Кривцова И.Е. Подход к выбору информативного признака в задаче идентификации программного обеспечения // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18, № 2. С. 278–285.
73. Рудина Т.Д. Разработка способа идентификации elf-файлов на основе нейронной сети. 2018. 52 с.
74. Салахутдинова, К.И. Лебедев И.С., Кривцова И.Е. Алгоритм градиентного бустинга деревьев решений в задаче идентификации программного обеспечения // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18, № 6.
75. Салахутдинова, К.И. Малков В.В., Кривцова И.Е. Сравнительный анализ подходов к идентификации программного обеспечения // Безопасность информационных технологий. 2019. Т. 26, № 2. С. 58–66.
76. Салахутдинова К.И. Повышение точности идентификации программного обеспечения путем использования аддитивного критерия Фишберна // Информационные технологии. 2019. Т. 25, № 10. С. 609–614.
77. Standard I. 24765-2017-I.I. Systems and software engineering--Vocabulary. 2017.
78. Benford F. The Law of Anomalous Numbers // Proceedings of the American Philosophical Society, 78. 1938. С. 551–572.
79. Fewster R.M. A simple explanation of Benford’s Law // Am. Stat. 2009. Т. 63, № 1. С. 26–32.
80. TESTING BENFORD’S LAW. File sizes in the Linux 2.6.39.2 source tree [Электронный ресурс]. URL: http://testingbenfordslaw.com/linux-filesizes.
81. Салахутдинова К.И., Лебедев И.С., Кривцова И.Е. Подход к выбору информативного признака в задаче идентификации программного обеспечения // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18, № 2. С. 278–285.
82. Salakhutdinova K.I. и др. Studying the Effect of Selection of the Sign and Ratio in the Formation of a Signature in a Program Identification Problem // Autom. Control

Comput. Sci. 2018. Т. 52, № 8. С. 1101–1104.

83. Смирнов Н.В., Дунин-Барковский И.. Курс теории вероятностей и математической статистики для технических приложений. М.: Наука, 1969. 511 с.
84. Куликов Е.И. Прикладной статистический анализ. М.: Горячая линия-Телеком, 2008. 463 с.
85. Сидоренко Е.В. Методы математической обработки в психологии. СПб.: Речь, 2010. 349 с.
86. Семенов В.А. Теория вероятностей и математическая статистика: Учебное пособие. Стандарт третьего поколения. СПб.: Питер, 2013. 192 с.
87. Каллан Р. Основные концепции нейронных сетей. Пер. с англ.— М.: Издательский дом «Вильямс», 2001. 291 с.
88. Электронный учебник по статистике [Электронный ресурс]. URL: http://statsoft.ru/home/textbook/modules/stneunet.html.
89. Кафтанников И.Л., Парасич А.В. Особенности применения деревьев решений в задачах классификации // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника». 2015. Т. 3, № 15. С. 26–32.
90. Freund Y., Schapire R. Experiments with a New Boosting Algorithm // Machine Learning: Proceedings of the Thirteenth International Conference. 1996. С. 148–156.
91. Дружков П.Н., Золотых Н.Ю., Половинкин А.Н. Реализация параллельного алгоритма предсказания в методе градиентного бустинга деревьев решений // Вестник ЮУрГУ. 2011. Т. 37, № 254. С. 82–89.
92. LightGBM GitHub [Электронный ресурс].
93. XGBoost GitHub [Электронный ресурс]. URL: https://github.com/dmlc/xgboost.
94. Китов В.В. Исследование точности метода градиентного бустинга со случайными поворотами // Статистика и экономика. 2016. Т. 4. С. 22–26.
95. CatBoost [Электронный ресурс]. URL: https://catboost.ai/docs/.
96. Коварцев А.Н., Даниленко А.Н. Алгоритмы и анализ сложности: учебник. Изд-во Сам. Самара, 2018. 128 с.

97. Подиновский В.В. Аксиоматическое решение проблемы оценки важности критериев в многокритериальных задачах // Современное состояние теории исследования операций. 1979. С. 117–149.
98. Анохин А.М. и др. Методы определения коэффициентов важности критериев // Автомат. и телемех. 1997. № 8. С. 3–35.
99. Фишберн П. Теория полезности для принятия решений. 1978. 352 с.
100. Фишберн П. Методы оценки аддитивных ценностей // Статистическое измерение качественных характеристик. 1972. С. 8–34.
101. Салахутдинова К.И. и др. Идентификации программного обеспечения в задаче аудита электронных носителей информации // Авиакосмическое приборостроение. 2019. № 9. С. 20–28.

Приложение 1. Значения информативности для 118 ассемблерных команд

$I(x)$	Ассемблерная команда	$I(x)$	Ассемблерная команда	$I(x)$	Ассемблерная команда
0,2024	cmpsb	0,2024	aaa	0,2025	rcl
0,2024	cmpsw	0,2024	les	0,2025	std
0,2024	esc	0,2024	daa	0,2025	sbb
0,2024	jc	0,2024	aas	0,2025	lahf
0,2024	jcxz	0,2024	aam	0,2025	ror
0,2024	jna	0,2024	ja	0,2025	ret
0,2024	jnae	0,2024	mul	0,2025	jne
0,2024	jnb	0,2024	js	0,2025	cmc
0,2024	jnbe	0,2024	loop	0,2025	popf
0,2024	jnc	0,2024	jge	0,2025	rcr
0,2024	jng	0,2024	jp	0,2025	adc
0,2024	jnge	0,2024	hlt	0,2025	jno
0,2024	jnl	0,2024	jl	0,2025	in
0,2024	jnle	0,2024	jns	0,2025	shl
0,2024	jnz	0,2024	loopne	0,2026	imul
0,2024	jpe	0,2024	iret	0,2026	sar
0,2024	jpo	0,2024	das	0,2026	sahf
0,2024	jz	0,2024	rep	0,2026	jmp
0,2024	lodsb	0,2024	jae	0,2026	xor
0,2024	lodsw	0,2024	idiv	0,2026	nop
0,2024	loopnz	0,2025	jbe	0,2026	and
0,2024	loopz	0,2025	jb	0,2026	jo
0,2024	movsb	0,2025	pushf	0,2026	je
0,2024	movsw	0,2025	not	0,2026	stc
0,2024	repe	0,2025	clc	0,2026	test
0,2024	repne	0,2025	rol	0,2026	push
0,2024	retn	0,2025	int	0,2026	retf
0,2024	sal	0,2025	jle	0,2027	out
0,2024	scasb	0,2025	jg	0,2027	cmp
0,2024	scasw	0,2025	sub	0,2027	inc
0,2024	stosb	0,2025	loope	0,2028	or
0,2024	stosw	0,2025	xlat	0,2029	xchg
0,2024	wait	0,2025	cld	0,203	pop
0,2024	cwd	0,2025	sti	0,203	add
0,2024	cbw	0,2025	shr	0,2032	lea
0,2024	div	0,2025	jnp	0,2036	call
0,2024	neg	0,2025	repnz	0,2044	mov
0,2024	into	0,2025	cli	0,2095	lock
0,2024	lds	0,2025	dec		
0,2024	aad	0,2025	repz		

Приложение 2. Числовые идентификаторы эталонных программ

Имя программы	Числовой идентификатор	Имя программы	Числовой идентификатор	Имя программы	Числовой идентификатор
acpid	1	autogen	15	beanstalkd	28
acpid	1	avahi-autoipd	16	beanstalkd	28
acpid	1	avahi-autoipd	16	bison	29
acpid	1	avahi-autoipd	16	bison	29
activity-log-manager	2	avahi-autoipd	16	bison	29
activity-log-manager	2	avahi-autoipd	16	bogofilter-bdb	30
activity-log-manager	2	avahi-daemon	17	bogofilter-bdb	30
aide	3	avahi-daemon	17	bonnie++	31
aide	3	avahi-daemon	17	brasero	32
aide	3	avahi-daemon	17	brasero	32
aide	3	avahi-daemon	17	brasero	32
amarok	4	b43-fwcutter	18	brasero	32
anacron	5	b43-fwcutter	18	brasero	32
anacron	5	b43-fwcutter	18	bsd-mailx	33
anacron	5	bacula-console	19	bsd-mailx	33
apmd	6	bacula-console	19	bsd-mailx	33
apmd	6	bacula-console	19	bsd-mailx	33
appstream-util	7	bacula-console	19	bsd-mailx	33
appstream-util	7	bacula-console-qt	20	bsd-mailx	33
apt	8	bacula-console-qt	20	ccache	34
apt	8	bacula-console-qt	20	ccache	34
apt	8	bacula-fd	21	ccache	34
aptitude	9	bacula-fd	21	ccache	34
aptitude	9	bacula-fd	21	ccache	34
ark	10	bacula-fd	21	ccache	34
ark	10	bacula-sd	22	cdparanoia	35
aspell	11	bacula-sd	22	cdparanoia	35
aspell	11	bacula-sd	22	cdrdao	36
aspell	11	bacula-sd+dfsg	23	cdrdao	36
aspell	11	bacula-sd+dfsg	23	ceph-fuse	37
at	12	bacula-sd+dfsg	23	ceph-fuse	37
at	12	bacula-traymonitor	24	ceph-fuse	37
at	12	bacula-traymonitor	24	checkbox-gui	38
at	12	baobab	25	checkbox-gui	38
attr	13	baobab	25	cheese	39
attr	13	baobab	25	cheese	39
authbind	14	bc	26	cheese	39
authbind	14	bc	26	cheese	39
authbind	14	bc	26	choqok	40
autogen	15	bdfresize	27	choqok	40
autogen	15	bdfresize	27	chrpath	41
autogen	15	autogen	15	chrpath	41

Продолжение таблицы

Имя программы	Числовой идентификатор	Имя программы	Числовой идентификатор	Имя программы	Числовой идентификатор
clamdscan	42	curl	50	wodim	62
clamdscan	42	curl	50	wodim	62
cmake	43	curl	50	xbrlapi	63
cmake	43	curl	50	xbrlapi	63
cmake	43	curl	50	xbrlapi	63
cmake	43	curl	50	xbrlapi	63
cmake	43	cvs	51		
cmake	43	cvs	51		
conntrack	44	cvs	51		
conntrack	44	cvs	51		
conntrackd	45	cvs	51		
conntrackd	45	cvsps	52		
corosync	46	cvsps	52		
corosync	46	cvsps	52		
corosync	46	cvsps	52		
corosync	46	dc	53		
corosync	46	dc	53		
corosync	46	dc	53		
crash	47	dc	53		
crash	47	finger	54		
crash	47	finger	54		
crash	47	gceph	55		
crash	47	gceph	55		
crash	47	genisoimage	56		
cron	48	genisoimage	56		
cron	48	genisoimage	56		
cron	48	genisoimage	56		
cron	48	iasl	57		
cron	48	iasl	57		
cron	48	jsvc	58		
cups-browsed	49	jsvc	58		
cups-browsed	49	nfct	59		
cups-browsed	49	nfct	59		
cups-browsed	49	radosgw	60		
cups-browsed	49	radosgw	60		
cups-browsed	49	radosgw	60		
curl	50	radosgw	60		
curl	50	radosgw	60		
curl	50	radosgw	60		
curl	50	rbd-fuse	61		
curl	50	rbd-fuse	61		
curl	50	rbd-fuse	61		
curl	50	wodim	62		
curl	50	wodim	62		

Приложение 3. Коды сравнения сигнатур с помощью библиотек градиентного бустинга деревьев решений

Таблица В.1 – LightGBM

Задаем входные данные

```
import pandas
import lightgbm
import numpy

traindata = pandas.read_csv('tradd.csv',header=None,sep=';')
testdata = pandas.read_csv('tsadd.csv',header=None,sep=';')
ytrain=traindata[0]
ytest=testdata[0]
xtrain=traindata.drop(0, axis=1)
xtest=testdata.drop(0, axis=1)
train = lightgbm.Dataset(xtrain,ytrain)
```

Использование библиотеки lightgbm и рассчет *Accuracy*

```
param = {
    'boosting_type': 'gbdt',
    'objective': 'multiclass',
    'metric': 'multi_logloss',
    'num_class': 64,
    'num_threads': 4,
    'learning_rate': 0.3,
    'max_depth': 7,
    'verbosity': 2
}
num_round = 1000
for learnrate in [0.3]:
    for i in [4]:
        param['learning_rate'] = learnrate
```

```
    param['max_depth'] = i
    model = lightgbm.train(param,train,num_round)
    preds = model.predict(xtest)
    result = []
    for j in range (len(preds)):
        result.append(preds[j].argmax())
    error_rate = numpy.sum(result != ytest)/ytest.shape[0]
    print("learnrate",learnrate,"max_depth",i," accuracy: ",1-error_rate)
```

Таблица В.2 – CatBoost

Задаем входные данные

```
train_data = []
test_data = []
train_labels = []
test_labels = []

with open("Iden/Train_data_ne_unif/sig_ob","r") as f:
    for line in f:
        train_data.append([int(x) for x in line.split()])

with open("Iden/Train_data/sig","r") as f:
    for line in f:
        test_data.append([int(x) for x in line.split()])

for x in range(len(test_data)):
    test_labels.append(test_data[x][0])
    del test_data[x][0]

for x in range(len(train_data)):
    train_labels.append(train_data[x][0])
    del train_data[x][0]
```

Использование библиотеки CatBoostClassifier и рассчет *Accuracy*

```
from catboost import Pool, CatBoostClassifier

pool = Pool(train_data, train_labels)
eval_pool = Pool(test_data, test_labels)
# Initialize CatBoostClassifier
model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.7,
    depth=2,
    loss_function='MultiClass',
    l2_leaf_reg = 1,
)
# Fit model
model.fit(pool, plot=True)
# Get predicted classes
preds_class = model.predict(test_data)
# Get predicted probabilities for each class
preds_proba = model.predict_proba(test_data)
# Get predicted RawFormulaVal
```

```
preds_raw = model1.predict(test_data, prediction_type='RawFormulaVal')
```

Рассчитываем *Accuracy*

```
tp = 0
fp = 0
for t in range(len(preds_class)):
    if test_labels[t] == preds_class[t]:
        tp += 1
    else:
        fp += 1
print("Accuracy = ", tp/(tp+fp))
```

Таблица В.3 – XGBoost

Задаем входные данные

```
from __future__ import division
import numpy as np
import xgboost as xgb
import pandas as pd

#get data fo further training and test
#label need to be 0 to num_class -1
traindata = np.genfromtxt('trje.csv',dtype='int', delimiter=';', converters={29: lambda x:int(x), 30: lambda x:int(x)-1})
testdata = np.loadtxt('tsje.csv',dtype='int', delimiter=';', converters={29: lambda x:int(x), 30: lambda x:int(x)-1})
sztr = traindata.shape
sztst = testdata.shape

train_X = traindata[:, :]
train_Y = traindata[:, 30]

test_X = testdata[:, :]
test_Y = testdata[:, 30]

xg_train = xgb.DMatrix(train_X, label=train_Y)
xg_test = xgb.DMatrix(test_X, label=test_Y)
```

Использование библиотеки xgb и расчет *Accuracy*

```
# setup parameters for xgboost
param = {}
# use softmax multi-class classification
param['objective'] = 'multi:softmax'
# scale weight of positive examples
param['eta'] = 0.3
param['max_depth'] = 2
param['nthread'] = 4
param['num_class'] = 63
param['silent'] = 1

watchlist = [(xg_train, 'train'), (xg_test, 'test')]
num_round = 1000
bst = xgb.train(param, xg_train, num_round, watchlist)
pred = bst.predict(xg_test)
error_rate = (1-np.sum(pred != test_Y) / test_Y.shape[0])*100
print("eta=",param['eta'],'accuracy = {}'.format(error_rate),' %')
```

Приложение 4. Свидетельство о регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019619363

Сравнение сигнатур исполняемых файлов

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Санкт-Петербургский институт информатики и автоматизации Российской академии наук (RU)***

Автор: ***Салахутдинова Ксения Иркиновна (RU)***

Заявка № **2019617199**

Дата поступления **14 июня 2019 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***16 июля 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016614206

Программа мониторинга Google Apps for Business

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Ильин Алексей Геннадьевич (RU), Первушин Алексей Олегович (RU), Юшковский Артем Викторович (RU), Катаева Валентина Алексеевна (RU), Павлов Кирилл Сергеевич (RU), Салахутдинова Ксения Иркиновна (RU)***

Заявка № **2015662892**
Дата поступления **28 декабря 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***18 апреля 2016 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016611975

Программа для поиска и анализа криптоконтейнеров с носителя информации

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Ильин Алексей Геннадьевич (RU), Первушин Алексей Олегович (RU), Юшковский Артем Викторович (RU), Катаева Валентина Алексеевна (RU), Павлов Кирилл Сергеевич (RU), Салахутдинова Ксения Иркиновна (RU)***

Заявка № **2015662891**
Дата поступления **28 декабря 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 февраля 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016614208

Программа для аудита событий информационной безопасности на основе модели MapReduce

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Ильин Алексей Геннадьевич (RU), Первушин Алексей Олегович (RU), Юшковский Артем Викторович (RU), Катаева Валентина Алексеевна (RU), Павлов Кирилл Сергеевич (RU), Салахутдинова Ксения Иркиновна (RU)***

Заявка № **2015662894**
Дата поступления **28 декабря 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***18 апреля 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016614048

Программа для криминалистического анализа IM ICQ и Jabber

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Салахутдинова Ксения Иркиновна (RU), Первушин Алексей Олегович (RU), Юшковский Артем Викторович (RU), Катаева Валентина Алексеевна (RU), Павлов Кирилл Сергеевич (RU), Ильин Алексей Геннадьевич (RU)***

Заявка № **2015663001**
Дата поступления **28 декабря 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***13 апреля 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016614207

Программа для стеганографического сокрытия информации в медиа файлах

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Ильин Алексей Геннадьевич (RU), Первушин Алексей Олегович (RU), Юшковский Артем Викторович (RU), Катаева Валентина Алексеевна (RU), Павлов Кирилл Сергеевич (RU), Салахутдинова Ксения Иркиновиа (RU)***

Заявка № **2015662893**
Дата поступления **28 декабря 2015 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***18 апреля 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Приложение 5. Копии актов внедрения

УНИВЕРСИТЕТ ИТМО

МИНОБРНАУКИ РОССИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный
исследовательский университет
информационных технологий,
механики и оптики» (Университет ИТМО)

Кронверкский проспект, д. 49, г. Санкт-Петербург,
Российская Федерация, 197101
тел.: (812) 232-97-04 | факс: (812) 232-23-07
od@mail.ifmo.ru | www.ifmo.ru

30.09.2019 № И-25/1213

УТВЕРЖДАЮ
Проректор по научной работе
д.т.н., профессор
_____ Никифоров В.О.
«___»_____2019г.

АКТ

об использовании результатов диссертационной работы
Салахутдиновой Ксении Иркиновны
«Методика идентификации исполняемых файлов на основе статического сбора характеристик дизассемблированного кода программ»
в учебном процессе университета

Настоящий акт составлен в том, что результаты диссертационной работы Салахутдиновой Ксении Иркиновны, а именно:

- методы формирования эталонных сигнатур программ и сигнатур идентифицируемых исполняемых файлов;
- методы сравнения сигнатур идентифицируемых исполняемых файлов с ранее сформированными эталонными сигнатурами программ;
- использование аддитивного критерия Фишберна в качестве постобработки результатов классификации.

используются факультетом БИТ (безопасности информационных технологий) Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики в учебном процессе при подготовке бакалавров по специальности 10.03.01 «Информационная безопасность» по дисциплинам «Организация и управление службой защиты информации» и «Теория вероятностей», а также при подготовке магистров по специальности 10.04.01 «Информационная безопасность» по дисциплинам «Методы цифровой обработки видеоизображений» и «Управление информационной безопасностью» при чтении курсов лекций, проведении практических и лабораторных работ.

Декан факультета БИТ,
к.т.н., доцент — Заколдаев Д.А.

УТВЕРЖДАЮ
Директор СПИИРАН
д.т.н., профессор РАН,
_____ Ронжин А.Л.
«__» октября 2019г.

АКТ

об использовании результатов диссертационного исследования
Салахутдиновой Ксении Иркиновны
«Методика идентификации исполняемых файлов на основе статического анализа характеристик дизассемблированного кода программ»

Комиссия в составе: председателя – заведующего лабораторией интеллектуальных систем, д.т.н., профессора Лебедева Ильи Сергеевича, членов: старшего научного сотрудника, к.т.н. Сухопарова Михаила Евгеньевича и младшего научного сотрудника Семенова Виктора Викторовича, составила настоящий акт в том, что ниже перечисленные научные результаты диссертационной работы Салахутдиновой Ксении Иркиновны:

• формирование сигнатур программ, используемых в дальнейшем в методе сравнения сигнатур на основе градиентного бустинга деревьев решений;

• метод сравнения сигнатур на основе машинного обучения (в частности, градиентного бустинга деревьев решений).

Внедрены в следующих научных проектах, выполненных или выполняемых в СПИИРАН:

• НИР АААА-А18-118110790076-5 «Теория и распределенные алгоритмы самоорганизации группового поведения агентов в автономной миссии», 2018-2019гг.

Использовалась реализация подхода внутреннего мониторинга, основанная на использовании алгоритма градиентного бустинга применительно к задаче идентификации внутренней среды автономного агента, т.е. отождествления того или иного исполняемого файла с некоторой известной программой, не основываясь на его целостности. Тем самым обеспечивалась поддержка мер, направленных на обеспечение механизма внутреннего мониторинга состояния автономного агента мультиагентной системы.

Председатель комиссии:
Заведующий лабораторией интеллектуальных систем, д.т.н., профессор _____ (подпись) И.С. Лебедев

Члены комиссии:
старший научный сотрудник, к.т.н. _____ (подпись) М.Е. Сухопаров

младший научный сотрудник _____ (подпись) В.В. Семенов

УТВЕРЖДАЮ
Первый заместитель директора
СПбФ АО «НПК «ТРИСТАН»
_____ И.Н. Соловьев
«26» ноября 2019г.

АКТ

об использовании результатов диссертационного исследования
Салахутдиновой Ксении Иркиновны
«Методика идентификации исполняемых файлов на основе статического анализа характеристик дизассемблированного кода программ»

Комиссия в составе: председателя – заместителя директора по науке, к.т.н. Сухопарова Михаила Евгеньевича, членов: заместителя директора, к.т.н., Гринько Сергея Васильевича и заместителя директора по программному обеспечению, к.т.н. Шахпароняна Артёма Павловича, составила настоящий акт в том, что нижеперечисленные научные результаты диссертационной работы Салахутдиновой Ксении Иркиновны:

- методы сравнения сигнатуры идентифицируемого исполняемого файла с эталонной сигнатурой программы;
- методика идентификации программного обеспечения на основе статических характеристик программного кода файла.

Используются в работе отдела проектирования и разработки программного обеспечения Санкт-Петербургского филиала АО «НПК «ТРИСТАН» для мониторинга состояния внутренних сетей и поиска несанкционированно установленного программного обеспечения с целью минимизации количества существующих уязвимостей системы и предотвращения возможных атак на отдельные устройства сети. Использование результатов диссертационного исследования способствует поддержанию установленных организационных мер по мониторингу и анализу устанавливаемого программного обеспечения на компьютеры пользователей и создает предпосылки для повышения уровня защищенности информации конфиденциального характера в процессе взаимодействия пользователей ЭВМ с автоматизированной системой организации.

Председатель комиссии:

заместитель директора по науке, к.т.н. _____ (подпись) М.Е. Сухопаров

Члены комиссии:

заместитель директора, к.т.н. _____ (подпись) С.В. Гринько

заместитель директора по ПО, к.т.н. _____ (подпись) А.П. Шахпаронян

Приложение 6. Публикации соискателя по теме Диссертации

В научных журналах, рекомендованных ВАК:

1. Салахутдинова К.И., Лебедев И.С., Кривцова И.Е. Подход к выбору информативного признака в задаче идентификации программного обеспечения // Научно-технический вестник информационных технологий, механики и оптики - 2018. - Т. 18. - № 2(114). - С. 278–285

2. Салахутдинова, К.И. Лебедев И.С., Кривцова И.Е., Сухопаров М.Е. Исследование влияния выбора признака и коэффициента (ratio) при формировании сигнатуры в задаче по идентификации программ // Проблемы информационной безопасности. Компьютерные системы. 2018. № 1. С. 136–141.

3. Салахутдинова, К.И. Лебедев И.С., Кривцова И.Е. Алгоритм градиентного бустинга деревьев решений в задаче идентификации программного обеспечения // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18, № 6.

4. Салахутдинова К.И., Малков В.В., Кривцова И.Е. Сравнительный анализ подходов к идентификации программного обеспечения // Безопасность информационных технологий - 2019. - Т. 26. - № 2. - С. 58-66

5. Салахутдинова К.И., Лебедев И.С., Кривцова И.Е., Анисимов А.С. Идентификации программного обеспечения в задаче аудита электронных носителей информации // Авиакосмическое приборостроение - 2019. - № 9. - С. 20-28

6. Салахутдинова К.И. Повышение точности идентификации программного обеспечения путем использования аддитивного критерия Фишберна // Информационные технологии -2019. - Т. 25. -№ 10. - С. 609-614

7. Бажаев Н., Давыдов А.Е., Кривцова И.Е., Лебедев И.С., Салахутдинова К.И. Подход к анализу состояния информационной безопасности беспроводной сети // Прикладная информатика - 2016. - Т. 11. - № 6(66). - С. 121-128

8. Кривцова И.Е., Салахутдинова К.И., Юрин И.В. Метод идентификации исполняемых файлов по их сигнатурам // Вестник Государственного университета

морского и речного флота имени адмирала С.О. Макарова - 2016. - № 1(35). - С. 215-224

В изданиях, индексируемых в международных базах цитирования Web of Science, Scopus:

9. Lebedev I.S., Korzhuk V., Krivtsova I., Salakhutdinova K., Sukhoparov M.E., Tikhonov D. Using Preventive Measures for the Purpose of Assuring Information Security of Wireless Communication Channels // Proceedings of the 18th Conference of Open Innovations Association FRUCT - 2016, pp. 167-173

10. Bazhayev N., Lebedev I.S., Krivtsova I.E., Sukhoparov M.E., Salakhutdinova K., Davydov A.E., Shaparenko I.M. Evaluation of the available wireless remote devices subject to the information impact // 10th IEEE International Conference on Application of Information and Communication Technologies, AICT 2016 - Conference Proceedings (Azerbaijan, Baku, 12-14 October 2016) - 2016, pp. 1-6

11. Lebedev I.S., Krivtsova I.E., Korzhuk V., Bazhayev N., Sukhoparov M.E., Pecherkin S., Salakhutdinova K. The Analysis of Abnormal Behavior of the System Local Segment on the Basis of Statistical Data Obtained from the Network Infrastructure Monitoring // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) - 2016, Vol. 9870, pp. 503-511

12. Krivtsova I.E., Lebedev I.S., Salakhutdinova K.I. Identification of Executable Files on the basis of Statistical Criteria//Proceedings of the 20th Conference of Open Innovations Association FRUCT, IET - 2017, pp. 202-208

13. Salakhutdinova K., Lebedev I.S., Krivtsova I.E., Bazhayev N., Sukhoparov M.E., Smirnov P.I., Markelov D.V., Davydov A.E., Pecherkin S., Kolcherin D.V., Shaparenko I.M., Iskanderov Y. A Frequency Approach to Creation of Executable File Signatures for their Identification//11th IEEE International Conference on Application of Information and Communication Technologies, AICT 2017 - Conference Proceedings (Moscow, 20-22 september 2017), IET - 2017, pp. 261-267.

14. Salakhutdinova, K.I. Krivtsova I.E., Lebedev I.S., Sukhoparov M.E. An Approach to Selecting an Informative Feature in Software Identification // Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 2018. С. 318–327.

15. Salakhutdinova K.I., Lebedev I.S., Krivtsova I.E., Sukhoparov M.E. Studying the Effect of Selection of the Sign and Ratio in the Formation of a Signature in a Program Identification Problem // Automatic Control and Computer Sciences - 2018, Vol. 52, No. 8, pp. 1101–1104

16. Semenov, Viktor V., Ilya S. Lebedev, Mikhail E. Sukhoparov and Kseniya I. Salakhutdinova. Application of an Autonomous Object Behavior Model to Classify the Cybersecurity State. Internet of Things, Smart Spaces, and Next Generation Networks and Systems. NEW2AN 2019, ruSMART 2019. Lecture Notes in Computer Science - 2019, Vol. 11660, pp. 104-112

В других научных журналах и изданиях:

17. Салахутдинова К.И., Овсяникова В.В., Трофимов А.А., Бессонова Е.Е., Ефремов А.А., Настека А.В. Анализ защищенности систем "Умный дом"//Региональная информатика (РИ-2014). XIV Санкт-Петербургская международная конференция «Региональная информатика (РИ-2014)». Санкт-Петербург, 29-31 октября 2014 г.: Материалы конференции. \ СПОИСУ. – СПб, 2014. - 2014. - С. 124

18. Ефремов А.А., Трофимов А.А., Настека А.В., Салахутдинова К.И., Овсяникова В.В. Защита управляющих сигналов в системе «Умный дом»//Сборник тезисов докладов конгресса молодых ученых. Электронное издание. – СПб: Университет ИТМО, 2015. – 2015

19. Овсяникова В.В., Настека А.В., Ефремов А.А., Салахутдинова К.И., Трофимов А.А. Защита системы «Умный дом» от программных сбоев//Сборник тезисов докладов конгресса молодых ученых. Электронное издание. – СПб: Университет ИТМО, 2015. – 2015

20. Druzhinin N.K., Salakhutdinova K.I. Identification of executable file by dint of individual feature//ISPIT-2015. International Conference on Information Security and Protection of Information Technology. St. Petersburg, Russia, November 5-6, 2015, IET - 2015, pp. 45-47

21. Кривцова И.Е., Салахутдинова К.И. Применение x2-критерия для идентификации elf-файлов. V Всероссийский конгресс молодых ученых//Сборник ВКМУ – 2016

22. Салахутдинова К.И., Кривцова И.Е. Условия применения критерия Пирсона для идентификации исполняемых файлов. Региональная информатика "РИ-2016" - 2016

23. Салахутдинова К.И., Дружинин Н.К. Идентификация исполняемых файлов по их ассемблерным командам // Научные работы участников конкурса «Молодые ученые Университета ИТМО» 2016 года. 2017.

24. Салахутдинова К.И., Лебедев И.С. Применение методов статистического анализа для идентификаций версий программного обеспечения удаленных автономных объектов транспортных систем //Информационная безопасность регионов России (ИБРР-2017). 2017.

25. Салахутдинова К.И., Лебедев И.С. Использование градиентного бустинга в задаче сравнения сигнатур программ //Сборник тезисов докладов конгресса молодых ученых. 2018. 136-141 с.

26. Салахутдинова, К.И. Обзор существующих подходов по аудиту электронных носителей информации // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. 2019.

Свидетельства о государственной регистрации программы для ЭВМ:

27. Ильин А.Г., Салахутдинова К.И., Юшковский А.В., Катаева В.А., Первушин А.О., Павлов К.С. Программа мониторинга Google Apps for Business. №2016614206, 18.04.2016.

28. Ильин А.Г., Салахутдинова К.И., Юшковский А.В., Катаева В.А., Первушин А.О., Павлов К.С. Программа для стеганографического сокрытия информации в медиа файлах. №2016614207,18.04.2016.

29. Ильин А.Г., Салахутдинова К.И., Юшковский А.В., Катаева В.А., Первушин А.О., Павлов К.С. Программа для поиска и анализа криптоконтейнеров с носителя информации. №2016611975, 15.02.2016.

30. Ильин А.Г., Салахутдинова К.И., Юшковский А.В., Катаева В.А., Первушин А.О., Павлов К.С. Программа для криминалистического анализа IM ICQ и Jabber. №2016614048, 13.04.2016.

31. Ильин А.Г., Салахутдинова К.И., Юшковский А.В., Катаева В.А., Первушин А.О., Павлов К.С. Программа для аудита событий информационной безопасности на основе модели MapReduce. №2016614208, 18.04.2016.

32. Салахутдинова К.И. Сравнение сигнатур исполняемых файлов. Свидетельство о государственной регистрации программы для ЭВМ №2019619363, 16.07.2019.