Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Уфимский государственный авиационный
технический университет»

На правах рукописи

ГАВРИЛОВ Григорий Николаевич

СИСТЕМА ОБНАРУЖЕНИЯ ВРЕДОНОСНЫХ ПРОГРАММ В ОПЕРАЦИОННОЙ СИСТЕМЕ (ОС) ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ (НА ПРИМЕРЕ ANDROID) С ПРИМЕНЕНИЕМ ИНТЕЛЛЕКТУАЛЬНЫХ ТЕХНОЛОГИЙ

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

Жернаков С.В.

Уфа – 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время мобильные устройства являются неотъемлемой частью повседневной жизни. Они используются для выполнения многочисленных операций и хранения личной информации, такой как мультимедиа, документы, пароли, контакты и т.д. По статистике Википедии, ОС Android работает на 64 % мобильных устройств. Широчайшие функциональные возможности и наличие личных данных служат причиной, по которой злоумышленники заинтересованы получить доступ к информации. ОС для мобильных устройств обладает хорошо организованными защитными механизмами, но имеет ряд уязвимостей, что позволяет вредоносным программам получить несанкционированный доступ к ней.

По данным статистики антивирусов, число вредоносных программ в ОС для мобильных устройств превысило 12 миллионов в 2014 году, что более чем в десять раз выше, чем в 2012 году [105]. Такой рост числа вредоносных программ обусловлен ростом популярности, функциональных возможностей и объема личной информации в процессе использования ОС для мобильных устройств. Антивирусные программы работают на основе хорошо изученных признаков вредоносных программ – сигнатур, которые хранятся в базе антивирусных сигнатур как эталоны и с которыми в дальнейшем осуществляется сравнение для последующего их обнаружения. Если в текущий момент времени сигнатура той или иной вредоносной программы отсутствует в антивирусной базе сигнатур, то антивирусная программа ее не обнаружит, и она может длительное время существовать в ОС для мобильных устройств. Процедура получения сигнатуры вредоносной программы требует затрат времени и тщательного анализа исследуемой программы с целью выявления ее признаков, а также необходимо добавить сигнатуру в базу сигнатур и распространить на все мобильные устройства. Сегодня с развитием сети Интернет скорость распространения различной информации и программного обеспечения значительно увеличилась, следовательно, новые или модифицированные вредоносные программы могут

распространиться за небольшое количество времени на множество ОС для мобильных устройств и нанести огромный ущерб. В связи с этим в диссертационной работе была поставлена задача разработать систему обнаружения вредоносных программ на основе интеллектуальных методов, которая на различных этапах анализирует поведение вредоносной программы и выявляет ее, что позволяет значительно повысить эффективность обнаружения вредоносных программ и их модификаций в ОС для мобильных устройств, адаптировать дополнительные средства защиты информации и своевременно локализовать угрозу, тем самым повысив уровень защищенности ОС для мобильных устройств в целом.

Объект исследования – обнаружение вредоносных программ в ОС для мобильных устройств (на примере Android).

Предмет исследования – методы и алгоритмы обнаружения вредоносных программ в ОС для мобильных устройств (на примере Android) на основе интеллектуальных технологий.

Цель работы

Повышение эффективности обнаружения вредоносных программ в ОС для мобильных устройств (на примере Android) путем разработки моделей и алгоритмов на основе интеллектуальных технологий.

Задачи исследования

Для достижения поставленной цели в работе были поставлены и решены следующие задачи:

1. Исследовать работу ОС для мобильных устройств Android с точки зрения защищенности, привести перечень угроз и уязвимостей, также рассмотреть существующие встроенные и сторонние средства защиты информации.

2. Разработать системные модели процесса функционирования системы обнаружения вредоносных программ, исследовать множество прикладных программ и сформировать перечень особенностей их поведения.

3. Разработать алгоритмы обнаружения вредоносных программ в ОС для мобильных устройств Android с применением интеллектуальных технологий.

4. Предложить архитектуру интеллектуальной системы обнаружения вредоносных программ, провести вычислительные эксперименты с целью оценки эффективности предложенных алгоритмов.

5. Разработать программное обеспечение исследовательского прототипа системы обнаружения вредоносных программ в мобильной ОС Android.

Методы исследования

В процессе исследования использовались теория вероятностей и методы математической статистики, методы иерархической кластеризации, метод к-средних, факторный анализ, кластерный анализ, дискриминантный анализ, нейросетевые методы, машина опорных векторов, аппарат нечеткой логики.

Научная новизна

Научная новизна работы заключается в следующем:

1. Предложена и обоснована экспериментальная выборка, которая позволяет описать поведенческий характер потенциальной вредоносной программы, на основе исследования множества прикладных программ, позволяющая осуществить их классификацию.

2. Разработаны системные модели процесса функционирования системы обнаружения вредоносных программ для ОС Android с учетом различных факторов на основе SADT-методологии и IDFE-технологий, позволяющие интегрировать систему обнаружения вредоносных программ с компонентами внутренних и внешних механизмов защиты ОС Android.

3. Разработан комплекс алгоритмов обнаружения вредоносных программ в ОС Android на основе интеллектуальных технологий, позволяющий с высокой точностью выполнять обнаружение вредоносных программ, отличающийся от классического сигнатурного метода динамическим анализом поведения вредоносной программы.

4. Предложена архитектура системы обнаружения вредоносных программ на основе предложенных алгоритмов, которая позволила обнаруживать как известные, так и модифицированные и новые вредоносные программы в ОС Android.

Практическая значимость

Практическая значимость разработанной системы обнаружения вредоносных программ заключается в применении комплекса данных алгоритмов с целью увеличения эффективности обнаружения вредоносных программ, а также в применении их в качестве дополнительного средства защиты к уже имеющимся методам обнаружения вредоносных программ в ОС для мобильных устройств Android.

Защищаемые положения

1. Результаты анализа защищенности ОС для мобильных устройств Android, встроенных и сторонних средств защиты информации, структуры прикладных программ и перечень угроз и уязвимостей.

2. Комплекс моделей функционирования системы обнаружения вредоносных программ в ОС для мобильных устройств Android и экспериментальная выборка, описывающая поведение двух типов прикладных программ.

3. Комплекс алгоритмов обнаружения вредоносных программ на основе машины опорных векторов и нечеткой логики.

4. Разработанный в виде программы исследовательский прототип системы обнаружения вредоносных программ в ОС для мобильных устройств Android.

Апробация результатов

Основные положения, представленные в диссертационной работе, докладывались и обсуждались на следующих конференциях:

- Международная научно-практическая конференция “Интеграционные процессы науки XXI века”, г. Стерлитамак, 2015 г.,

- Международная научно-практическая конференция “Научные перспективы XXI века”, г. Нефтекамск, 2015 г.,

- XIII Международная научно-практическая конференция “Научные перспективы XXI века. Достижения и перспективы нового столетия”, г. Новосибирск, 2015 г.,

- XV Международная научно-практическая конференция “Научное обозрение физико-математических и технических наук в XXI веке”, г. Москва, 2015 г.,

- XVII Международная научно-практическая конференция “Актуальные вопросы развития инновационной деятельности в новом тысячелетии”, г. Новосибирск, 2015 г.,

- Международная молодежная научная конференция “XXII Туполевские чтения (школа молодых ученых)”, г. Казань, 2015 г.,

- XVI Международная научно-техническая конференция “Проблемы техники и технологии телекоммуникаций”, г. Уфа, 2015 г.,

- IX Всероссийская молодежная научная конференция “Мавлютовские чтения”, г. Уфа, 2015 г.,

- XV Международная научная конференция “Перспективы направления развития современной науки”, г. Москва, 2016 г.,

- VIII Международная научно-практическая конференция “Актуальные проблемы науки XXI века”, г. Москва, 2016 г.,

- XII международная научная-техническая конференция «Актуальные проблемы электронного приборостроения», г. Новосибирск, 2016 г.

Публикации

По теме диссертации опубликовано 17 научных статей и тезисов докладов, из них 6 статьи в изданиях, рекомендованных ВАК.

Структура работы

Диссертационная работа включает введение, четыре главы основного материала, заключение, приложения А, Б, В, Г и список литературы. Работа изложена на 196 страницах машинописного текста, список литературы включает 118 наименований.

ГЛАВА 1 АНАЛИЗ СОВРЕМЕННОГО СОСТОЯНИЯ ЗАЩИЩЕННОСТИ ОС ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

1.1 Обзор современного состояния защищенности ОС для мобильных устройств

Мобильные устройства являются неотъемлемой частью нашей повседневной жизни, количество персональных данных, которые они хранят, постоянно увеличивается. В отличие от персональных компьютеров, ОС для мобильных устройств развиваются более динамично. По имеющимся данным Википедии (рисунок 1.1), ОС Android установлена на 64 % устройств [103, 118]. Учитывая, что доля устройств с ОС Android на рынке постоянно увеличивается, то и число потенциальных пользователей современных мобильных устройств будет продолжать неуклонно расти [114, 116, 117]. На диаграмме приведена статистика использования ОС для мобильных устройств.

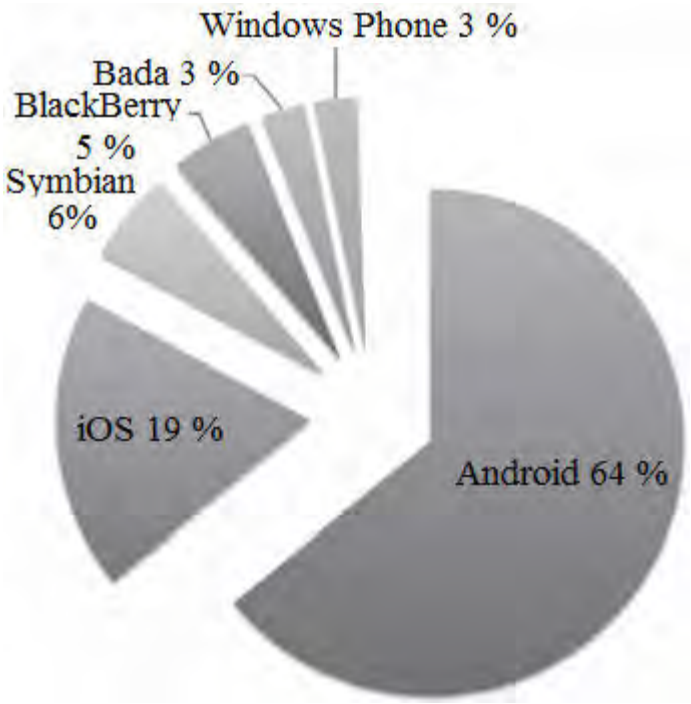


Рисунок 1.1 – Статистика использования ОС для мобильных устройств

При этом область защиты информации не всегда поддерживается разработчиком и успевает вслед за развитием мобильных устройств. В настоящее время они широко используются для выполнения множества операций, таких как работа с программой банк-клиент, почтой, документами, облачными сервисами, фото, видео и т.д. Мобильное устройство представляет собой банк персональной информации, при потере контроля над которым можно утратить все персональные данные, финансы и т.д., например, десятки тысяч пользователей Сбербанка стали жертвами мошенничества. Как заявил информационному агентству FlashNord источник в МВД, пострадали 20-30 тысяч человек. Все пострадавшие использовали мобильные устройства различных марок на базе ОС Android. Троянская программа, которая относится к семейству вредоносных программ Backdoor.AndroidOS.Obad, похищала денежные средства с “привязанных” к телефонным номерам карт Сбербанка. Пользователи долго оставались в неведении, так как программа блокировала смс-сообщения о снятии средств со счета. Ущерб составил миллионы рублей [66].

За последние два года число вредоносных программ, нацеленных на ОС для мобильных устройств, выросло более чем в 10 раз и составило 12 миллионов в 2014 году [106].

В 2014 году антивирусные продукты “Лаборатории Касперского” заблокировали 6 167 233 068 вредоносных атак на компьютеры и мобильные устройства пользователей. Отражено 1 363 549 атак на ОС Android.

Решения, разработанные и внедренные “Лабораторией Касперского”, отразили 1 432 660 467 атак, осуществлявшихся интернет-ресурсами, находящимися в разных странах мира.

За данный период времени было обнаружено в ОС для мобильных устройств:

- 4 643 582 вредоносных установочных пакета;
- 295 539 новых вредоносных программ;
- 12 100 мобильных банковских троянских программ [107].

Несколько лет назад на мобильных устройствах вредоносных программ практически не существовало, так как их функциональные возможности были весьма ограниченны, а также отсутствовали сервис и услуги, которые они сейчас предлагают пользователю. Разработчики ОС для мобильных устройств с момента разработки учитывали в своих продуктах максимальный уровень защищенности от вредоносных программ. ОС для мобильных устройств не позволяли вредоносным программам “захватывать” управление устройством.

В настоящее время ситуация изменилась, в первую очередь благодаря расширению возможностей самих мобильных устройств. Современное мобильное устройство – это интегрально-модульная система: полноценный рабочий инструмент, центр развлечений и средство управления личными финансами. Рост функциональных возможностей мобильных устройств требует от злоумышленников разработки новых вредоносных программ, а также изощренных способов распространения и заражения.

На рисунке 1.2 изображен рост числа мобильных троянских программ и их модификаций по данным “Лаборатории Касперского” [107].

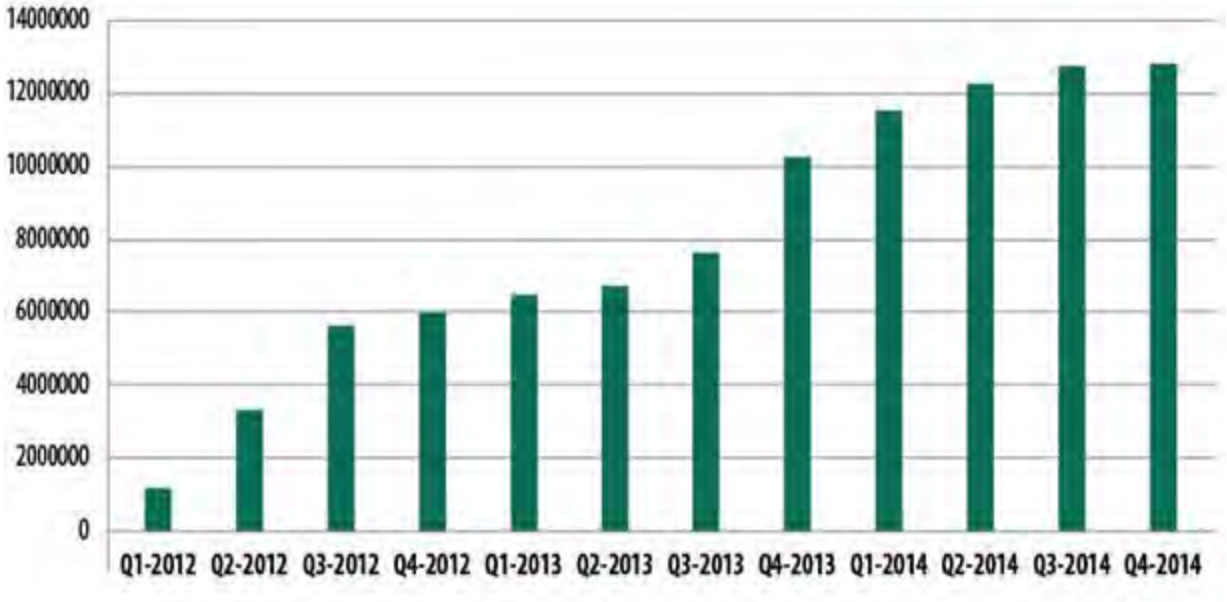


Рисунок 1.2 – Количество обнаруженных вредоносных программ

С 2012 года число мобильных вредоносных программ выросло более чем на порядок и к концу 2014 года, таким образом, превысило 12 миллионов. Из зависимости, приведенной на рисунке 1.2, видно, что рост вредоносных программ значительно увеличивается по мере увеличения популярности, числа пользователей и расширения круга услуг и сервисов.

Рисунок 1.3 – Распределение мобильных вредоносных программ по функциям

Показательно меняется распределение вредоносных программ по типам (рисунок 1.3): троянская программа, которая способна отправлять смс-сообщения, и эксплоиты, которые используют дефект алгоритма написанного кода, уступают “дорогу” рекламным вредоносным и троянским программам, направленным на банковские продукты. При этом уменьшение доли вредоносных программ какого-либо типа не свидетельствует о том, что они не применяются и не разрабатываются, о чем говорит рост общего числа вредоносных программ [28].

При работе с мобильным устройством пользователь не всегда задумывается о возможной потере важной для него информации: потеря устройства, его продажа и т.д. – все эти и другие факторы не являются критическими по

сравнению с намеренными атаками на мобильное устройство, так как в этом случае риск потери конфиденциальной информации увеличивается.

На фоне анализа общей ситуации в сфере информационной безопасности, касающейся вредоносных программ на ОС для мобильных устройств, можно сделать вывод, что рост популярности таких ОС и их функциональных возможностей, расширение функционала приводит к количественному и качественному росту вредоносных программ, о чем свидетельствует приведенная статистика.

1.2 Обзор архитектуры и компонентов ОС для мобильных устройств

ОС для мобильных устройств представляет собой в значительной степени “усеченное” по функционалу ядро ОС Linux, а также встроенное прикладное и системное программное обеспечение.

С точки зрения архитектуры она представляет собой полный программный стек, который состоит из четырех уровней, описанных в таблице 1.1 [97].

Таблица 1.1 – Четыре уровня архитектуры

Название	Описание
Базовый уровень (ядро ОС Linux)	Уровень абстракции между аппаратным уровнем и программным стеком
Набор библиотек и среда исполнения	Обеспечивает основной базовый функционал для работы программ, содержит виртуальную машину и базовые библиотеки Ява, необходимые для запуска программ
Уровень каркаса программ	Обеспечивает разработчикам доступ к интерфейсу программирования, предоставляемый компонентами системы уровня библиотек
Уровень программ	Набор предустановленных базовых системных программ

В основании архитектуры лежит “усеченное” ядро ОС Linux, которое служит промежуточным уровнем между аппаратным и программным обеспечением. Оно обеспечивает функционирование всей системы и предоставляет системные службы ядра: управление памятью, энергосистемой и процессами, обеспечивает безопасность, работу с сетью и драйверами.

Уровнем выше располагается набор библиотек и среда исполнения. Библиотеки выполняют следующие функции:

- предоставляют реализованные алгоритмы для вышележащих уровней;
- обеспечивают поддержку файловых форматов;
- осуществляют кодирование и декодирование информации, например, мультимедиакодеки;
- выполняют организацию отображения графики и т.д.

На рисунке 1.4 представлена архитектура ОС для мобильных устройств (на примере ОС Android).

Библиотеки реализованы на языке C/C++ и скомпилированы под определенное аппаратное обеспечение мобильного устройства, с которым они поставляются производителем в предустановленном виде. Перечень и описание основных типов библиотек представлены в таблице 1.2.

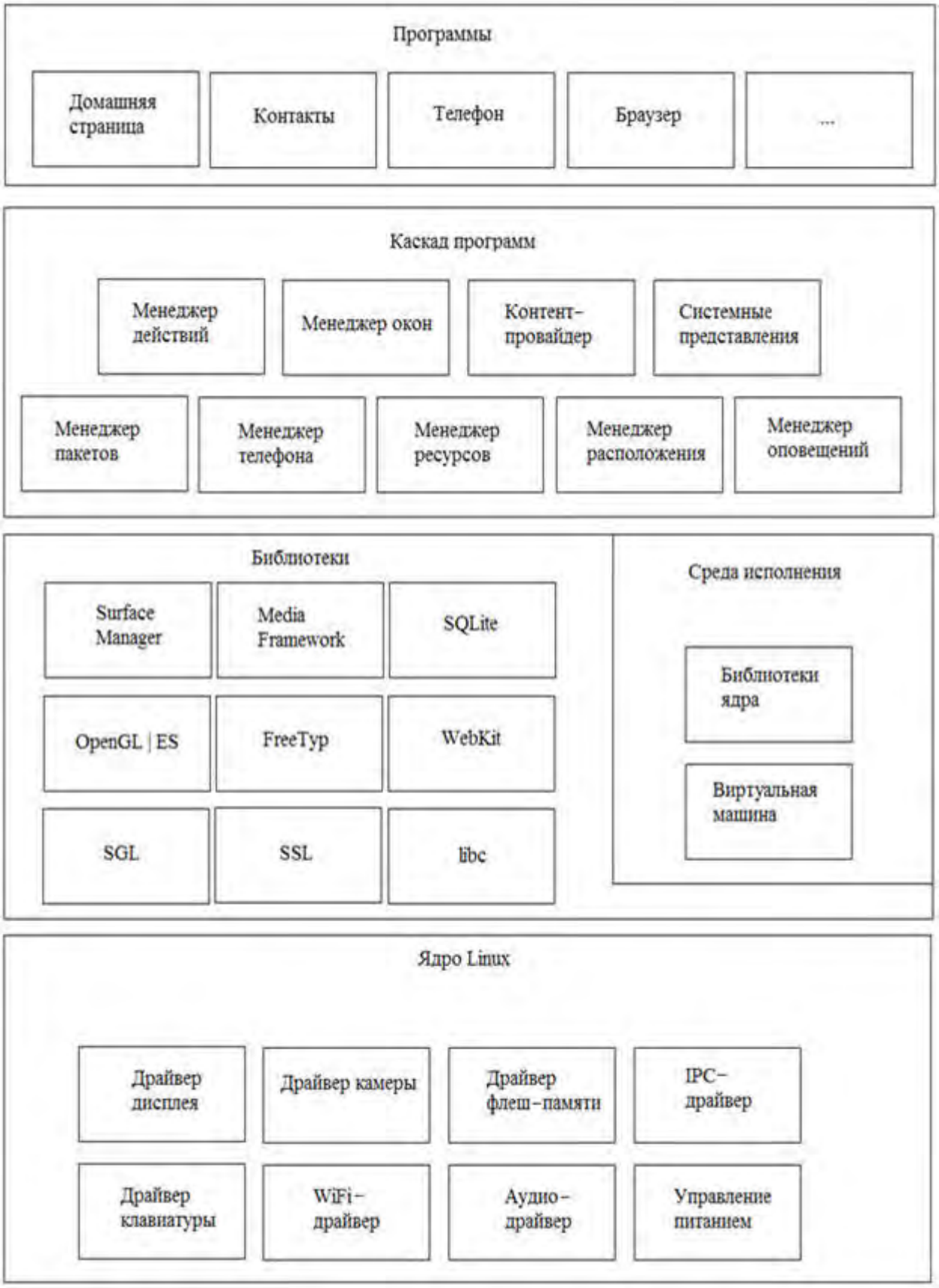


Рисунок 1.4 – Архитектура ОС (на примере ОС Android)

Таблица 1.2 – Перечень основных библиотек

Название	Описание
1	2
Менеджер поверхностей	Представляет собой композитный менеджер окон. Поступающие команды по организации отображения графики собираются в закадровый буфер, накапливаются в нем, составляя некую композицию, и затем выводятся на экран мобильного устройства. Данная библиотека позволяет создавать различные эффекты, прозрачность окон и плавные переходы
Каркас мультимедиа Ява-программ	Библиотеки, реализованные на базе кодека PacketVideo OpenCORE. Необходимы для записи и воспроизведения аудио- и видеосодержимого, а также для вывода изображений
SQLite	Легковесная и производительная реляционная система управления базами данных, используется в качестве основного “движка” для работы с базами данных
3D-библиотеки	Используются для оптимизированного на высоком уровне отображения 3D-графики, при технической возможности задействуется аппаратное ускорение. Библиотеки реализованы на основе интерфейса программирования OpenGL|ES. OpenGL|ES (OpenGL for Embedded Systems) – подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах
FreeType	Библиотека для работы с битовыми картами, для растеризации (перевод изображения векторной формы в пиксели) шрифтов и осуществления операций над ними
LibWebCore	Библиотеки ядра встроенного браузера

Окончание таблицы 1.2

1	2
SGL (графическое ядро “Skia”)	Используется для работы с 2D-графикой
SSL	Библиотеки для поддержки одноименного криптографического протокола
Libc	Стандартная библиотека языка С, а именно ее реализация, настроенная для работы на устройствах на базе ОС Linux

Среда исполнения включает в себя библиотеки ядра, за счет которых обеспечивается значительная часть низкоуровневой функциональности, которая доступна библиотекам ядра языка Ява, и виртуальную машину, позволяющую запускать программы. Каждая программа запускается в своем экземпляре виртуальной машины, таким образом обеспечивается изоляция работающих программ в ОС для мобильных устройств друг от друга. Для исполнения программы на виртуальной машине выполняется компиляция Ява-классов в исполняемые файлы с расширением .dex с помощью инструмента dx, входящего в состав среды разработки программ (Android SDK). DEX (Dalvik EXecutable) – формат исполняемых файлов для виртуальной машины, оптимизированный для использования минимального объема памяти. При использовании IDE Eclipse и плагина, который имеет название “средства разработки программ”, компиляция классов Ява в формат .dex происходит автоматически.

Архитектура среды исполнения реализована таким образом, что функционирование программ осуществляется строго в рамках окружения виртуальной машины, что позволяет защитить ядро от возможной угрозы со стороны других ее составляющих. Поэтому если сработает код с ошибками или вредоносное программное обеспечение, то они не смогут вывести из строя ОС.

Уровнем выше располагается каркас программ, особенности его архитектуры позволяют любой программе использовать уже реализованные

возможности других программ, к которым разрешен доступ. В таблице 1.3 приведено описание компонентов каркаса программ.

Таблица 1.3 – Компоненты в составе каркаса программ

Название	Описание
Богатый и расширяемый набор представлений	Компонент, который необходим для создания визуальных составляющих программ (списков, текстовых полей, таблиц)
Контент-провайдеры (источники данных)	Осуществляет управление данными, которые одни программы открывают для других, чтобы последние могли их задействовать для своей работы
Менеджер ресурсов	Содержит функции для доступа к ресурсам программы (строкам, значениям, графике, макетам пользовательского интерфейса и др.)
Менеджер оповещений	Реализует возможность для программ отображать уведомления для пользователя в строке состояния
Менеджер действий	Управляет жизненными циклами программ, сохраняет историю работы с действиями, предоставляет систему навигации по действиям
Менеджер местоположения	Позволяет программам периодически получать обновленные данные о текущем географическом положении мобильного устройства

Каркас программ предоставляет в распоряжение программам дополнительный вспомогательный функционал, который реализует принцип многократного использования компонентов программ в рамках политики безопасности.

Уровень программ является самым близкорасположенным к пользователю уровнем. На данном уровне пользователь взаимодействует со своим мобильным

устройством, а также на нем представлен набор установленных базовых программ.

Для инсталляции прикладных программ пользователь может воспользоваться встроенным облачным сервисом с каталогом программ, который позволяет осуществлять их покупку и инсталляцию. Чтобы инсталлировать прикладную программу в ОС для мобильных устройств, создается файл с расширением .apk, который содержит исполняемые файлы и вспомогательные компоненты (файлы, содержащие данные и ресурсы). После инсталляции прикладной программы каждая программа функционирует в своем собственном изолированном экземпляре виртуальной машины [108, 110].

Большинство мобильных устройств содержит следующие аппаратные составляющие:

- процессор, производительность которого ограничена, чтобы сократить тепловыделение;
- чипы памяти;
- чип накопителя и – в ряде моделей – слот для дополнительного накопителя;
- аккумуляторная батарея;
- графический процессор;
- звуковой процессор и динамики;
- сенсоры, в числе которых акселерометры, компас, светочувствительные датчики и т.д.;
- GPS-приемник;
- антенна Wi-Fi и – в ряде моделей – сотовая антенна;
- чип Bluetooth;
- FM-тюнер;
- камера (одна и более).

Таким образом, выполнив анализ архитектуры и компонентов ОС для мобильных устройств (на примере ОС Android), можно сделать вывод, что она

представляет собой сложно организованную структуру, работающую по определенным алгоритмам. В основе лежит “усеченное” ядро ОС Linux, которое управляет работой ОС для мобильных устройств. Каркас программ, программы, среда исполнения и библиотеки являются компонентами и образуют архитектуру ОС для мобильных устройств. Совместно с ядром они выполняют весь функционал, а также обеспечивают безопасность информации встроенными средствами ее защиты. Сложность системы обусловлена наличием широкого функционала и высоких требований к работе мобильного устройства.

1.3 Обзор структуры и составляющих прикладных программ ОС для мобильных устройств

Прикладная программа ОС для мобильных устройств (на примере ОС Android) имеет архитектуру, представленную на рисунке 1.5.

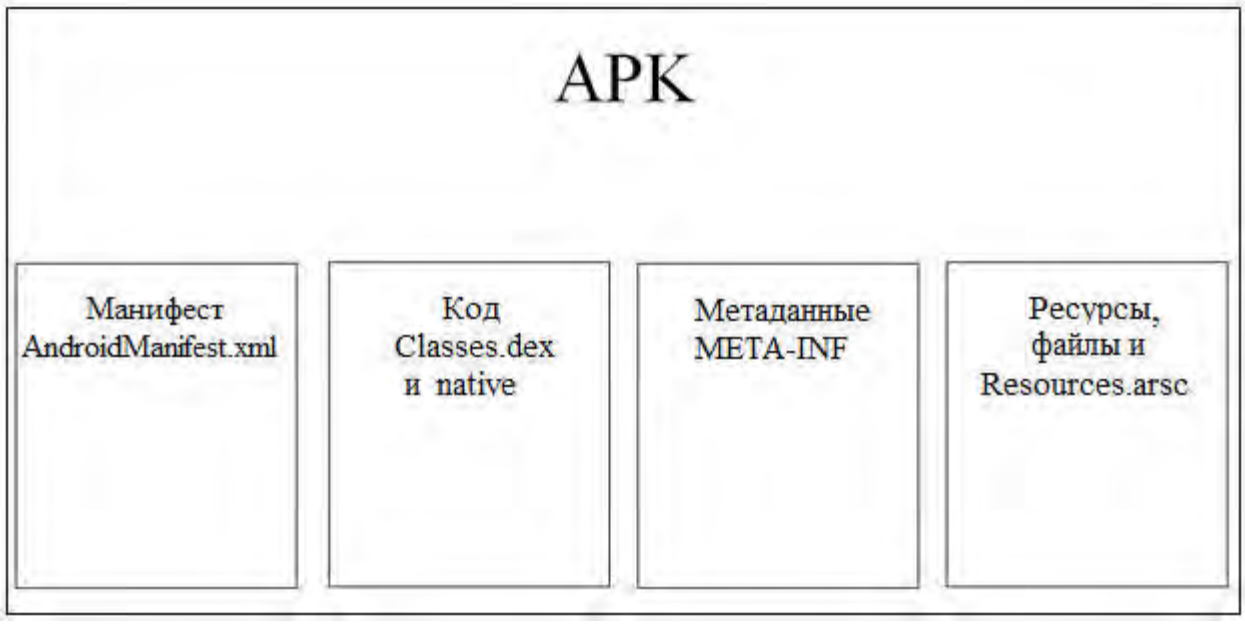


Рисунок 1.5 – Структура прикладной программы ОС Android

Прикладная программа представляет собой APK архив, который включает в себя следующие компоненты [104]:

ФАЙЛ МАНИФЕСТ состоит из перечня аппаратных и программных требований, необходимых для запуска и работы прикладной программы. Он имеет название AndroidManifest.xml и содержит всю информацию о прикладной программе, которая при обращении предоставляет ее ОС. Каждая прикладная программа имеет свой файл манифест, который представляет собой XML-код. Он необходим, чтобы:

- предоставить имя Ява-пакета прикладной программы и в последующем выступать в качестве уникального идентификатора;
- описывать компоненты, входящие в состав прикладной программы;
- содержать список всех необходимых разрешений для обращения к программным и аппаратным составляющим;
- объявлять разрешения, которые прикладные программы обязаны иметь для взаимодействия с компонентами данной программы;
- выполнять прочие запросы на библиотеки и т.д.

В связи с важностью данного файла необходимо рассмотреть подробно его структуру и составляющие. На рисунках 1.6–1.8 представлена общая структура файла манифест.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest/>
<uses-permission/>
<permission/>
<permission-tree/>
<permission-group/>
<instrumentation/>
<uses-sdk/>
<uses-configuration/>
<uses-feature/>
<supports-screens/>
```

Рисунок 1.6 – Первая часть структуры файла манифест

Элемент manifest является корневым элементом файла манифест. В своем составе имеет четыре атрибута [3, 45]:

- xmlns:android – данный атрибут определяет пространство имен;
- Package – определяет уникальное имя пакета прикладной программы, которое регламентируется при его создании;
- android:versionCode – содержит внутренний номер версии, необходимый для сравнения версий прикладных программ;
- android:versionName – содержит номер версии прикладной программы, который видим для пользователя.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package= "ru.grigoriygavrilov.helloandroid"
android:versionCode="1"
android:versionName="1.0">
```

Рисунок 1.7 – Пример атрибутов manifest

Элемент permission объявляет разрешения, которые необходимы для ограничения доступа к определенным компонентам или функциональности данной прикладной программы. В данном элементе содержится описание прав, которые запрашивают другие прикладные программы для получения доступа к искомой программе, а также прикладная программа может защитить свои собственные компоненты (службы, контент-провайдеры) путем использования разрешений. Она использует любое из системных разрешений или объявленных другими программами и может определить свои собственные разрешения [33].

Элемент uses-permission осуществляет запрос списка разрешений, которые будут предоставлены прикладной программе для корректного функционирования. Они предоставляются и запрашиваются на этапе инсталляции прикладной программы.

Например, следующие разрешения:

- INTERNET – запрос прав на доступ к Интернету;
- READ_CONTACTS – запрос прав на чтение, но не на запись данных из адресной книги пользователя;

- WRITE_CONTACTS – запрос прав на запись, но не на чтение данных из адресной книги пользователя;
- RECEIVE_SMS – запрос прав на обработку входящих смс-сообщений;
- ACCESS_COARSE_LOCATION – запрос прав на определение приблизительного местонахождения при помощи вышек сотовой связи или точек доступа Wi-Fi;
- ACCESS_FINE_LOCATION – запрос прав на точное определение местонахождения при помощи навигации и т.д.

Элемент permission-tree объявляет базовое имя для дерева разрешений, но не само разрешение, а только пространство имен, в которое могут быть помещены дальнейшие разрешения.

Элемент permission-group определяет имя для набора логически связанных разрешений. Данный элемент объявляет исключительно категорию, в которую могут быть помещены разрешения.

Элемент instrumentation объявляет объект instrumentation, который дает возможность контролировать взаимодействие программ в ОС.

Элемент uses-sdk указывает требуемую совместимость прикладной программы с указанной версией ОС.

Элемент uses-configuration указывает требуемую для прикладной программы аппаратную и программную конфигурацию мобильного устройства.

Элемент uses-feature объявляет определенные параметры, относящиеся к функциональности, которые необходимы для работы прикладной программы. Следовательно, прикладная программа, которая не имеет соответствующих функциональных возможностей, не будет инсталлирована.

Элемент supports-screens задает разрешение экрана, необходимое для корректного отображения прикладных программ в ОС для мобильного устройства.

Элемент application – один из основных элементов файла манифест, содержащий описание компонентов прикладных программ, доступных в пакете:

стили, значок и др. Содержит дочерние элементы, которые объявляют каждый из компонентов, входящих в состав прикладных программ.

Элемент activity объявляет активность (действие) и содержит множество других атрибутов, определяющих разрешения, ориентацию экрана и т. д.

Элемент intent-filter определяет типы действий, с помощью которых вызываются компоненты прикладной программы (выбрать фотографию, сделать звонок и т.д.) и на которые могут ответить “деятельность” (окна программы), сервис или приемник действий. Фильтр действий объявляет возможности его родительского компонента: что могут сделать “деятельность” или служба и какие типы рассылок получатель может обработать. Фильтр действий предоставляет для компонентов-клиентов возможность получения действия объявляемого типа, фильтрации тех, которые не имеют значимости для компонента, а также содержит дочерние элементы [72]:

- элемент Action – добавляет действие к фильтру действий;
- элемент Category – определяет категорию компонента, которую должно обработать действие;
- элемент Data – добавляет спецификацию данных к фильтру действий.

Элемент meta-data определяет пару “имя-значение” для элемента дополнительных произвольных данных, которыми можно обеспечить родительский компонент.

Элемент activity – alias-это аналог (псевдоним) Activity.

Элемент service объявляет службу как один из компонентов программы.

Элемент receiver объявляет приемник широковещательных действий как один из компонентов программы. Приемники широковещательных действий дают возможность прикладной программе получить действия, которые переданы системой или другими программами, даже когда другие компоненты программы не работают.

Элемент provider объявляет контент-провайдера (источник данных) для управления доступом к базам данных. Элемент grant-uri-permission является дочерним элементом для provider. Он определяет, для кого можно предоставить

разрешения на подмножества данных контент-провайдера. Элемент path-permission – дочерний элемент для provider. Определяет путь и требуемые разрешения для определенного подмножества данных в пределах поставщика оперативной информации.

```
<application>
 <activity>
  <intent-filter>
   <action/>
   <category/>
   <data/>
   </intent-filter>
   <meta-data/>
 </activity>
 <activity-alias>
   <intent-filter>
   <action/>
   <category/>
   <data/>
   </intent-filter>
   <meta-data/>
</activity-alias>
<service>
  <intent-filter>
    <action/>
    <category/>
    <data/>
    </intent-filter>
    <meta-data/>
</service>
<receiver>
   <intent-filter>
   <action/>
    <category/>
    <data/>
   </intent-filter>
   <meta-data/>
 </receiver>
 <provider>
   <grant-uri-permission>
         <path-permission/>
   <meta-data/>
  </provider>
  <uses-library/
 </application>
</manifest>
<manifest>
```

Рисунок 1.8 – Вторая часть структуры файла манифест

Элемент uses-library определяет общедоступную библиотеку, с которой должна быть скомпонована программа, и указывает системе на необходимость включения кода библиотеки в загрузчик классов для пакета программы. Каждый проект связан по умолчанию с библиотеками, в которые включены основные пакеты для сборки программы. Однако некоторые пакеты находятся в отдельных библиотеках, которые автоматически не компонуются с программой. Если программа использует пакеты из этих библиотек или других, от сторонних разработчиков, то необходимо сделать явное связывание с этими библиотеками, файл манифест обязательно должен содержать отдельный элемент uses-library:

КОД ПРОГРАММЫ, который будет выполняться в ОС, он представлен в виде байт-кода, скомпилированного для среды исполнения,

МЕТАДАННЫЕ, содержащие в себе контрольную сумму, электронную цифровую подпись для контроля целостности программы,

РЕСУРСЫ И ФАЙЛЫ ПРОГРАММЫ – это картинки, мультимедиафайлы и прочие ресурсы.

Код программы состоит из следующих компонентов:

- Ява-классов, которые представляют собой подклассы основных классов из среды разработки (Android SDK) (View, Activity, ContentProvider, Service, BroadcastReciever, Intent);
- Ява-классов, у которых нет родителей в среде разработки (Android SDK).

Каждая программа запускается в своем собственном процессе, поэтому программа изолирована от других запущенных программ, и некорректно работающая программа не может беспрепятственно навредить другим программам. Тем не менее одной из основных особенностей программы ОС является возможность использовать компоненты других программ, если они предоставляют соответствующие права. Например, необходим компонент для отображения текста, похожий компонент уже реализован в другой программе, следовательно, можно задействовать и использовать уже реализованный компонент другой программы. В данном случае программа не копирует

необходимый код, не создает ссылку на него, а делает запрос на исполнение части кода другой программы, в которой присутствует нужный для работы компонент.

Процесс инсталляции прикладной программы включает в себя следующие этапы:

1. Поиск прикладной программы. Прикладная программа может быть получена из облачного сервиса либо из неизвестного источника, а также она может попасть на устройство через персональный компьютер и прочие носители информации.

Существует три способа инсталляции прикладных программ в ОС для мобильных устройств:

1) посредством встроенной системной программы, то есть пакетного установщика. Прикладная программа инсталлируется с помощью встроенного системного пакетного установщика;

2) через облачный сервис или какой-либо другой источник сети Интернет. Программа устанавливается из облачного сервиса с каталогом программ;

3) через команду “adb install”. Прикладная программа инсталлируется с применением команды “adb install”, которую используют разработчики для инсталляции программ из персонального компьютера.

2. Инсталлируемая прикладная программа запрашивает список разрешений у пользователя, осуществляется проверка аппаратных и программных возможностей устройства, которые необходимы для ее работы в ОС для мобильных устройств.

3. Выполняется инсталляция кода прикладной программы в ОС для мобильных устройств. Прикладная программа начинает функционировать в изолированной среде, каждой программе присваивается свой каталог и пользователь.

Таким образом, выполнив обзор и анализ структуры прикладных программ, а также процесса инсталляции прикладных программ в ОС для мобильных устройств, можно сделать вывод, что прикладная программа произвольно не запускается и не функционирует, не имея соответствующих прав и программно-

аппаратных ресурсов со стороны ОС. Ключевым моментом данного процесса является пользователь, который предоставляет разрешение на запрашиваемые ресурсы и права, необходимые для работы прикладной программы, а также файл манифест, который содержит все требования, необходимые для ее работы. Следовательно, данный файл требует детального анализа для составления полной схемы функционирования прикладной программы, в частности определения, что она собой представляет, какое воздействие на ОС для мобильных устройств оказывает и какие действия выполняются данной прикладной программой в процессе функционирования.

1.4 Обзор встроенных и сторонних механизмов защиты в ОС для мобильных устройств

В ОС для мобильных устройств с открытым исходным кодом необходима надежная архитектура безопасности и программное обеспечение, так как она содержит в себе большой объем инструментов, сервисов и личной информации. В ОС Android реализована многоуровневая система безопасности, которая обеспечивает гибкость, необходимую для открытой платформы и обеспечивает защиту для всех пользователей. В системе реализован контроль безопасности с точки зрения как разработчиков, так пользователей.

Одна из важных функций безопасности, реализованная в ОС для мобильных устройств – это песочница, которая позволяет существенно уменьшить функциональные возможности ядра ОС и тем самым в какой-то степени обезопасить ее в целом.

Модель безопасности ОС для мобильных устройств использует функции безопасности, реализованные в ядре ОС Linux. Она является многопользовательской ОС, а ее ядро может разделять пользовательские ресурсы друг от друга так же, как они изолирует процессы. В ОС Linux один пользователь не может получить доступ к файлам другого пользователя, если не предоставлено соответствующее разрешение. Каждый процесс выполняется с неким

идентификатором: используется идентификатор пользователя – UID и группы – GID пользователя, процесс не запустится, если не установлен ID пользователя или ID группы, то есть так называемые SUID и SGID биты на соответствующий исполняемый файл. На рисунке 1.9 представлена модель безопасности современной ОС для мобильных устройств (на примере ОС Android). Она имеет хорошо организованную систему безопасности как на программном, так и на аппаратном уровнях. В данной системе реализованы и используются современные информационные технологии и множество других перспективных методов обеспечения защиты информации на устройстве, описанные ниже.

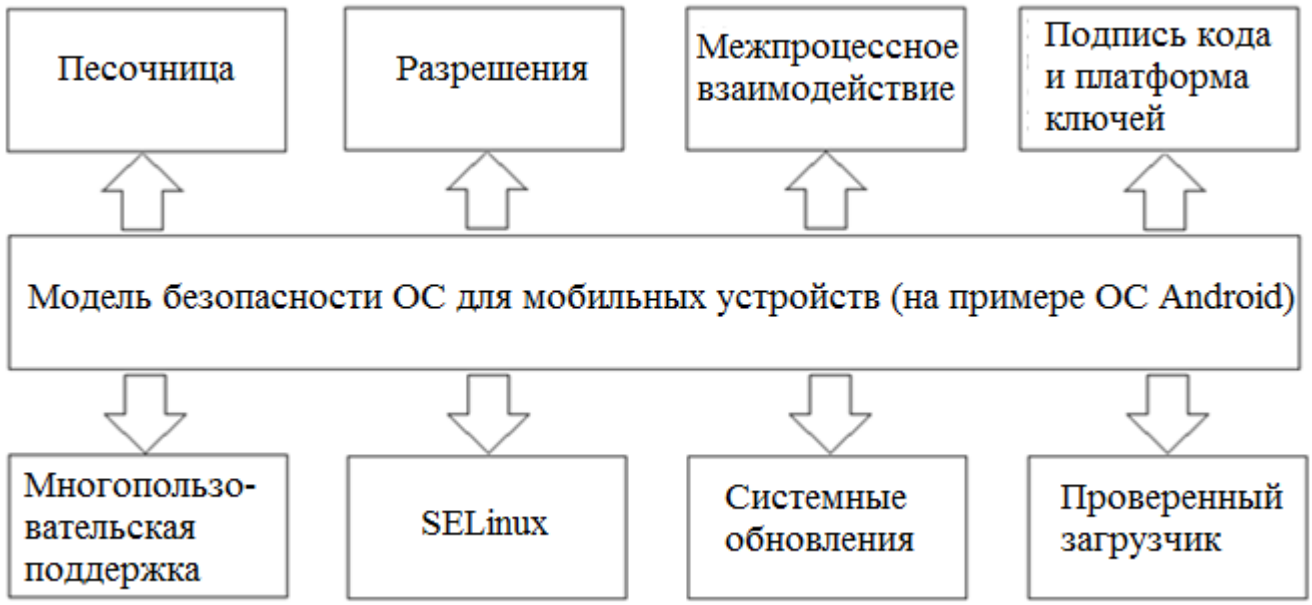


Рисунок 1.9 – Модель безопасности (на примере ОС Android)

Модель безопасности ОС для мобильных устройств включает в себя следующие 8 модулей:

1. Песочница.

Во время инсталляции программ, в ОС каждой программе по умолчанию присваиваются уникальные пользовательские идентификаторы ID (UID) и ID группы (GID), каждой программе соответствует свой пользователь. Имя пользователя имеет формат “app_x”, а идентификаторы пользователя вычисляют по формуле **(**Process.FIRST_APPLICATION_UID + x)

Process.FIRST_APPLICATION_UID равен 10000. Данные идентификаторы программы не изменяются. Список установленных программ хранится в файле packages.list, расположенном в директории “/data/system/”. У каждой программы существует своя домашняя директория, например, “/data/data/package_name”, где package_name – имя пакета, например, com.ex.ex1. Имя пакета задается в свойстве пакета, в файле AndroidManifest.xml. Данная папка является внутренним хранилищем, директорией, где программа хранит все свои личные данные, и к которой разработчики программы получают доступ посредством функций Context.getFilesDir() или Context.getDir(). У папки права доступа определены как drwxr-x--x, то есть только владелец и пользователи, входящие в группу владельцев, имеют полный доступ к ней. Каждая программа определена как уникальный пользователь, это означает, что она, по умолчанию, не имеет доступа к информации о других программах. Кроме того, на уровне ядра уникальные UID и GID каждой программы используются для разделения доступа к ресурсам, таким как память и процессор. Таким образом, на уровне ядра для каждой программы создается своя собственная песочница – среда, позволяющая ограничить действия программы и обезопасить систему. ОС автоматически присваивает уникальный UID, так называемый идентификатор программы, для каждой программы при инсталляции и выполняет эту программу как отдельный запущенный UID процесс. Помимо функционала, описанного выше, каждая программа получает специальный каталог, к которому только она имеет разрешение на чтение и запись.

Таким образом, программы помещаются в изолированную среду как на уровне процесса, так как выделяется отдельный процесс для каждой программы, так и на уровне файлов выделяется отдельный каталог для каждой программы. Механизм песочницы в мобильной ОС основан на присвоении каждой программе уникального пользовательского идентификатора UID и идентификатора группы GID, каждой программе соответствует свой уникальный непривилегированный пользователь.

2. Система разрешений.

Программы изолируются, получают доступ только к собственным файлам и к любым общедоступным ресурсам в ОС для мобильных устройств. Она может присвоить дополнительные права – разрешения, чтобы программа могла расширить свой функционал и задействовать дополнительные функции и ресурсы, необходимые для ее функционирования. Разрешения могут контролировать доступ к драйверам, интернет–соединению, данным или сервисам, а также к другим важным функциям.

Устанавливаемые программы запрашивают список разрешений через файл AndroidManifest.xml. Во время инсталляции программы ОС считывает список разрешений, и пользователь принимает решение, присваивать права программе или нет. После того как права предоставлены, они повторно не запрашиваются, будут присвоены программе по умолчанию. Есть несколько уровней доступа программы, они зависят от того, насколько далеко может зайти программа в процессе работы [115].

3. Фреймворк межпроцессного взаимодействия.

Программы имеют возможность выполнять запрос системных сервисов для того, чтобы получить доступ к ресурсам ОС, например, к данным навигации, а системные сервисы имеют возможность предоставить такую информацию программам. Вследствие того, что программы и системные сервисы исполняются в разных процессах для организации взаимодействия в ОС, существует механизм обмена информацией между процессами. Данный механизм также необходим программам из каталога программ, для организации взаимодействия между собой. В ОС для мобильных устройств обмен сигналами и данными между процессами организован с помощью такой функции, как фреймворк межпроцессного взаимодействия, но в некоторых случаях используются сокеты домена Unix. Все остальные способы взаимодействия между процессами реализованы с использованием фреймворка межпроцессного взаимодействия – он предоставляет возможность синхронно и асинхронно запрашивать методы удаленных объектов так, будто они локальные, обмениваться файловыми дескрипторами между

процессами, получать автоматическое оповещение в случае, если фреймворк межпроцессного взаимодействия определенного процесса был прерван, и т.д. Взаимодействие между процессами организовано по синхронной клиент-серверной модели. Клиент инициирует соединение и ждет ответа со стороны сервера, то есть взаимодействие между клиентом и сервером происходит последовательно. Это позволяет разработчику вызывать удаленные методы, как будто они находятся в одном процессе.

Схема взаимодействия процессов через фреймворк межпроцессного взаимодействия представлена на рисунке 1.10: программа “клиент”, которая выполняется в процессе A, получает доступ к функциональности, реализованной в сервисе, который выполняется в процессе B.

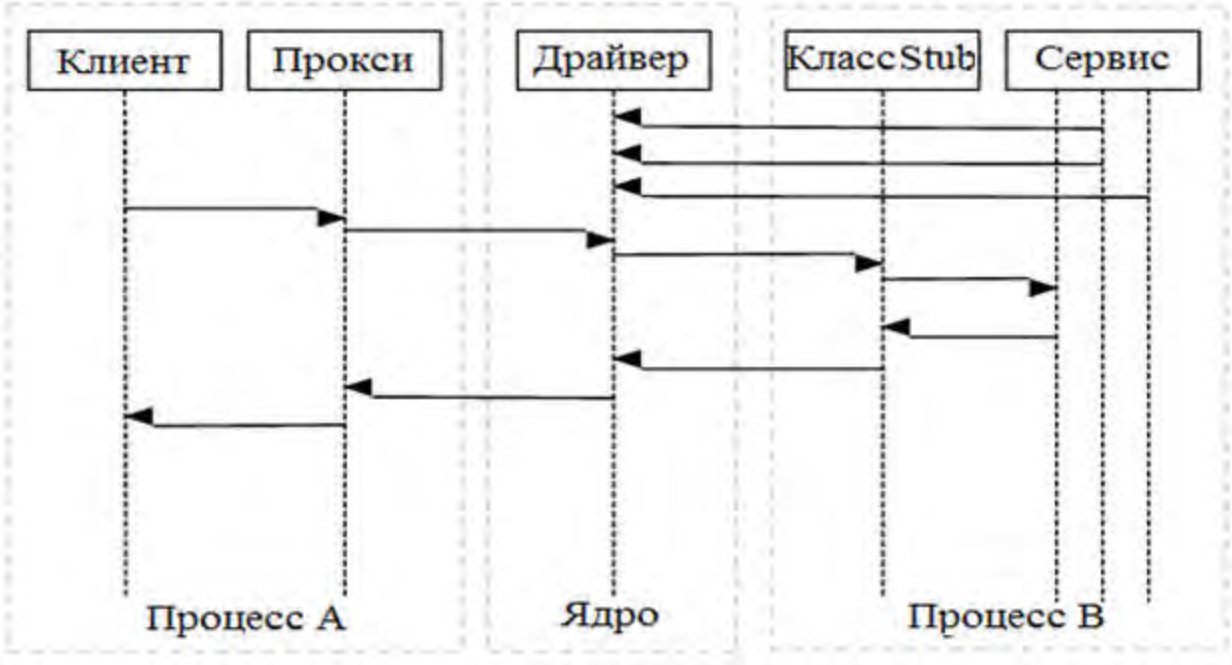


Рисунок 1.10 – Схема функционирования системы межпроцессорного взаимодействия

Все взаимодействия между клиентом и сервером в рамках фреймворка межпроцессного взаимодействия выполняются через специальный драйвер устройства ОС Linux, который находится в директории “/dev/binder”, при помощи класса, который называется Stub.

4. Подпись кода и платформа ключей.

Все сторонние и встроенные системные программы в ОС для мобильных устройств должны быть подписаны их разработчиком; так как это APK файлы с расширением формата Ява пакета JAR, то метод подписания кода используется на основе JAR подписания [22]. ОС использует подпись APK для того, чтобы убедиться, что обновления для программы приходят от того же автора, и для того, чтобы установить доверительные отношения между программами. Обе эти функции безопасности реализуются путем сравнения сертификатов подписи. Встроенные системные программы подписываются номером платформы ключей. Различные компоненты системы могут обмениваться ресурсами и работать внутри того же процесса, если они подписаны тем же ключом платформы.

5. Многопользовательский режим.

Данный режим предоставляет возможность использовать устройство нескольким пользователями и при этом у каждого присутствует свое отведенное пространство, ресурсы, которые недоступны остальным.

6. Мандатное управление доступом (SELinux).

Функция SELinux вносит дополнительный уровень безопасности в ОС для мобильных устройств, называемый “мандатное управление доступом”. Данный уровень конфигурируется с помощью общесистемных политик безопасности, он обеспечивает дополнительные возможности управления тем, как пользователи и программы получают доступ как к собственным, так и предоставляемым системой данным, причем это делается в фоновом режиме для пользователей. В техническом плане данная функция позволяет создавать политики, определяющие виды взаимодействий, разрешенные и не разрешенные для каждого процесса в рамках общего контекста безопасности. SELinux настроена на работу в факультативном режиме, в котором не обеспечивается применение политик, а нарушения только заносятся в журнал [96].

7. Системные обновления.

8. Проверенный загрузчик.

В настоящее время существует множество антивирусных программ [99]:

- AhnLab V3 Mobile;
- Anguanjia;
- Antiy AVL for Android;
- Avast Mobile Security;
- Avira Antivirus Security;
- Bitdefender Mobile Security;
- ESET Mobile Security;
- G Data Internet Security;
- Kaspersky Internet Security;
- Qihoo 360 Antivirus;
- Quick Heal Total Security;
- Trend Micro Mobile Security.

Антивирусная программа работает на основе сигнатурного анализа, следовательно, если вредоносные программы отсутствуют в базе антивирусных сигнатур или если базы сигнатур не обновлялись по какой-либо причине, то они не будут обнаружены. Сам процесс обнаружения, выделения и добавления сигнатуры в базы сигнатур антивирусных программ кропотлив и занимает много времени [75].

В целом ОС Android имеет организованную модель безопасности, в которой реализованы современные информационные технологии зашиты информации, но любая информационная система, даже самая современная, нуждается в модернизации систем защиты, и со временем злоумышленники находят уязвимости и пути обхода систем защиты.

В частности, используется способ защиты от вредоносных программ на основе системы разрешений, которые устанавливаемая программа запрашивает на этапе инсталляции, но не каждый пользователь анализирует список запроса необходимых функций.

Следовательно, ОС подвержена воздействию уязвимостей со стороны новых вредоносных программ. Новые вредоносные программы, модернизированные и

отсутствующие в базе сигнатур антивирусной программы, могут получить несанкционированный доступ и оказать негативное воздействие на ОС для мобильных устройств. Статистика свидетельствует о значительном приросте количества вредоносных программ, что является одним из основополагающих факторов для постановки вопроса о разработке современного средства обнаружения новых вредоносных программ, которое будет иметь возможность без наличия в базе сигнатур антивирусных программ, анализируя поведенческий характер программы, осуществлять обнаружение вредоносных программ.

1.5 Обзор и анализ угроз и уязвимостей в ОС для мобильных устройств

Архитектура ОС для мобильных устройств имеет интегрально-модульную структуру, а также встроенные средства защиты, которые функционируют таким образом, что все программы имеют ограниченные права доступа и не имеют связи с защищенными данными других программ [8].

Угрозой для мобильных ОС являются специальные программы-эксплойты, которые позволяют получить права доступа привилегированного пользователя, что в результате дает им возможность выполнять все возможные действия на аппаратном и программном уровнях: редактировать системные файлы, управлять работой аппаратной составляющей мобильного устройства, создавать и редактировать ОС без каких-либо ограничений. Данный тип наиболее опасных вредоносных программ способен снабдить злоумышленника всей информацией о пользователе, устройстве, местоположении и позволяет управлять мобильным устройством удаленно [57]. Злоумышленники используют этот вид угроз, а также уязвимости [81] для реализации комплексных атак и получения прав доступа с наивысшим приоритетом, чтобы беспрепятственно устанавливать программы без разрешения пользователя. Для разработки данной вредоносной программы требуется много времени, специалист высокой квалификации, также наличие актуальной уязвимости в алгоритме ОС. Приведенные выше факторы обусловливают низкую популярность данного типа вредоносных программ.

В целом вредоносные программы не используют так называемые эксплойты. Они вводят пользователя в заблуждение и пытаются спровоцировать его на выполнение необходимых действий: предоставить вредоносной программе все возможные разрешения на использование необходимых для ее функционирования ресурсов. Таким образом, предоставив вредоносной программе требуемые ей возможности, ничего не подозревающий пользователь добровольно дает последней разрешение на совершение вредоносных действий. Такого типа программы могут находиться как в известных источниках – облачных сервисах, так и в любом другом сервисе.

Одним из важных условий в системе безопасности ОС для мобильных устройств является система разрешений, которая представляет собой одну из уязвимостей. При инсталляции любой программы в ОС пользователю предоставляется список всех разрешений на программные и аппаратные сервисы, которые будут доступны программе согласно ее назначению. После этапа инсталляции программа получает возможность выполнять свои действия с предоставленными функциями без участия пользователя, а также в фоновом режиме. Вследствие этого пользователь не будет осведомлен о ее действиях. Система разрешений реализована с целью обеспечения безопасности работы пользователя, а также предоставляет возможность контроля устанавливаемых программ, однако большинство пользователей чаще всего не осведомлены и по этой причине не обращают внимания на список запрашиваемых функций. Отмеченная уязвимость является одной из самых серьезных, так как на данном этапе можно предотвратить инсталляцию программы и избежать всех возможных убытков, которые устанавливаемая вредоносная программа может повлечь за собой вследствие функционирования.

Угрозу также может представлять использование модифицированной ОС для мобильных устройств, которая может быть модифицирована любым пользователем для своих нужд. В такие ОС могут быть встроены вредоносные программы. Также когда цифровой подписью эталонного образа системы “подписывается” программа, она получает те же права, что и сама система, в

которой она работает. В рамках ОС Android с открытым исходным кодом подписи для образов являются приватными, поэтому такой сценарий возможен, например, в случае похищения соответствующей подписи. Подобный способ заражения применялся вредоносной программой Android.SmsHider, которая могла в фоновом режиме для пользователей, использующих модифицированные ОС для мобильных устройств, инсталлировать содержащуюся в ней троянскую программу из файла с расширением APK [95].

Встроенные системные программы также имеют уязвимости. Уязвимости встроенного браузера позволяют вредоносным программам выполнить код и получить доступ к защищенным данным браузера [105].

Уязвимостью также является открытость ОС для мобильных устройств. Так как текст кода находится в открытом доступе, он может активно использоваться злоумышленниками, чтобы находить уязвимости и ошибки с последующим их использованием для своих целей. Среди основных уязвимостей выделяют:

- возможность устанавливать программы как из официального каталога программ, так из любого другого доступного источника. Программы, которые находятся в доверенном облачном сервисе или источнике, проходят проверку антивирусной программой. Поэтому вероятность появления вредоносных программ в данном типе источника меньше, чем в неизвестном,
- возможность любому пользователю и разработчику размещать в официальном каталоге программ свои разработки. ОС Android является открытой, и любой пользователь может создавать и размещать свои труды в официальном облачном сервисе, что позволяет злоумышленникам свободно распространять вредоносные программы,
- отсутствие системных обновлений со стороны разработчика.

Важную роль играет человеческий фактор: вредоносные программы распространяются через рекламу в других программах с использованием хитро составленных рекламных фраз (“Необходимо срочно обновить систему”, “Ваша версия программы устарела”, “Немедленно установите обновление” и т. д.), а также при помощи рассылок спама посредством смс-сообщений. Например,

Android.Crusewind функционирует таким образом: пользователь получает смс-сообщение, информирующее о том, что ему необходимо обновить настройки доступа к сети Интернет, и также в сообщении содержится ссылка. Затем, перейдя по ссылке, пользователь получает файл в виде вредоносной программы.

Существуют следующие типы вредоносных программ, приведенные в таблице 1.4.

Таблица 1.4 – Типы вредоносных программ

Вредоносная программа	Описание
1	2
Троянская программа, отправляющая смс-сообщения	Программы данного типа отправляют смс-сообщения с повышенной тарификацией на короткие номера, таким образом часть стоимости данных сообщений поступает злоумышленнику на счет. Подобные программы практически ничем не отличаются друг от друга, кроме как незначительными изменениями в интерфейсе и разнообразием коротких номеров, на которые будет выполняться отправка смс-сообщений. В основном программы такого типа распространяются под видом популярных программ, таких как браузер Opera Mini, ICQ, Skype, Angry Birds и т. п., при этом используется соответствующая иконка, которая присуща данной популярной программе. В качестве примера можно привести вредоносную программу семейства Android.SmsSend [112]

Продолжение таблицы 1.4

1	2
Троянские программы	Сбор конфиденциальной информации о пользователе, добавление закладок в браузер, выполнение различных команд, поступающих от злоумышленников, отправка смс-сообщений, инсталляция других программ и т. д. Чтобы реализовать возможность инсталляции программ, не вызывая подозрений со стороны пользователя, необходимы права привилегированного пользователя, права, с которыми работает ядро системы. Например, Android.Gongfu, Android.Wukong, Android.DreamExploid, Android.Geinimi, Android.Spy и пр.
Коммерческие программы-шпионы	Данные программы используются для слежки за действиями пользователей: выполняют перехват входящих и исходящих смс-сообщений, звонков, записей на диктофон, отслеживание местоположения устройства, сбор данных из браузера и т. д. Учитывая тот факт, что большинство таких программ требуют первоначальной настройки и ручной инсталляции, они представляют собой существенную угрозу, так как после инсталляции на устройство работают в фоновом режиме и не создают ярлык среди других программ. Обнаружить их присутствие можно только по определенным признакам, в том числе зайдя в системное меню со списком программ или проанализировав системные процессы

Окончание таблицы 1.4

1	2
Рекламные модули	Программы, имеющие в своем составе сервисный модуль, который функционирует в фоновом режиме – после закрытия самой программы, и отображает рекламу в области уведомлений. Данный тип рекламных сообщений часто используется для публикации сообщений, замаскированных под системные, например, “Требуется срочное обновление системы” или “Появилась новая версия программы”, что может послужить путем распространения вредоносных программ. В облачном сервисе содержится большое количество программ с таким подходом к отображению рекламы. Например, Adware.Airpush,Adware.Leadbolt, Adware.Startapp [113]
Прочие	Вредоносные программы, которые сочетают в себе различные функции, описанные выше

Таким образом, одной из главных проблем безопасности ОС для мобильных устройств (на примере Android) является человеческий фактор, так как в большинстве отмеченных уязвимостей требуется непосредственное разрешение пользователя на запуск и функционирование вредоносной программы. Какой бы защищенной она ни была, человеческий фактор играет ключевую роль в распространении вредоносных программ на ОС для мобильных устройств. Систему разрешений можно расширить, добавив дополнительную степень контроля программ, чтобы потенциальные пользователи имели возможность увидеть, какие функции может задействовать та или иная программа. Если будет внедрена дополнительная система анализа разрешений, отображающая необходимость выполнения программами того или иного действия, а также позволяющая самому пользователю принимать решение, анализировать ее действия, это позволит значительно повысить уровень безопасности ОС для мобильных устройств в целом [114].

1.6 Выводы по первой главе. Цели и задачи исследования

1. Рассмотрены с точки зрения функционирования компоненты и архитектура ОС для мобильных устройств (на примере ОС Android), представляющая собой “усеченное” ядро ОС Linux, которое управляет мобильным устройством.

2. Показано, что вследствие роста функциональных возможностей ОС для мобильных устройств увеличивается количество вредоносных программ и их модификаций в ОС для мобильных устройств, о чем свидетельствует приведенная статистика.

3. Рассмотрены структура и процесс инсталляции прикладных и системных программ в ОС для мобильных устройств (на примере ОС Android), таким образом, ключевыми понятиями являются файл с названием “манифест” и система разрешений.

4. Установлено, что встроенные функции модели безопасности ОС для мобильных устройств, такие как песочница, система разрешений, межпроцессное взаимодействие, подпись кода и платформы ключей и т.д., – а также антивирусные программы обладают существенными недостатками, в частности система разрешений.

5. Установлено, что ОС для мобильных устройств имеет ряд уязвимостей, таких как система разрешений, встроенные программы и т.д., а также может подвергаться угрозе со стороны вредоносных программ.

На основании вышеизложенного сформулируем цель диссертационной работы – повышение эффективности обнаружения вредоносных программ в ОС для мобильных устройств (на примере Android) путем разработки моделей и алгоритмов на основе интеллектуальных технологий.

Для достижения данной цели были поставлены следующие задачи:

1. Исследовать работу ОС для мобильных устройств Android с точки зрения защищенности, привести перечень угроз и уязвимостей, также рассмотреть существующие встроенные и сторонние средства защиты информации.

2. Разработать системные модели процесса функционирования системы обнаружения вредоносных программ, исследовать множество прикладных программ и сформировать перечень особенностей их поведения.

3. Разработать алгоритмы обнаружения вредоносных программ в ОС для мобильных устройств Android с применением интеллектуальных технологий.

4. Предложить архитектуру интеллектуальной системы обнаружения вредоносных программ, провести вычислительные эксперименты с целью оценки эффективности предложенных алгоритмов.

5. Разработать программное обеспечение исследовательского прототипа системы обнаружения вредоносных программ в мобильной ОС Android.

ГЛАВА 2 РАЗРАБОТКА МОДЕЛЕЙ ФУНКЦИОНИРОВАНИЯ СИСТЕМЫ ОБНАРУЖЕНИЯ ВРЕДОНОСНЫХ ПРОГРАММ В ОС ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

2.1 Построение функциональных моделей IDEF0, IDEF1X работы системы обнаружения вредоносных программ в ОС для мобильных устройств

Система обнаружения вредоносных программ в ОС для мобильных устройств представляет собой решение, реализованное в виде программы (на уровне ОС), функционирующей по определенным правилам, инструкциям и требованиям. Отобразим ее функционирование на контекстной диаграмме согласно назначению, задачам и функциям [2, 87, 89]. Соответствующая контекстная диаграмма A0 модели IDEF0 начального уровня показана на рисунке 2.1.

Рисунок 2.1 – Контекстная диаграмма A0 работы модели системы обнаружения вредоносных программ в ОС Android

На вход подаются компоненты программы, содержащие в себе файлы androidmanifest.xml, classes.dex, и информация о системных вызовах, которые представляют собой материал для производства выхода. Управляющими стрелками служат правила, условия и требования, согласно которым регулируется деятельность и порядок работы системы обнаружения вредоносных программ в ОС для мобильных устройств. На выходе находятся результаты классификации машиной опорных векторов (ok, virus) и нечеткой логикой (в процентах и в присвоении категории программе (безопасная, подозрительная и опасная)), получаемые в результате работы системы обнаружения вредоносных программ. Программные средства, аппаратные ресурсы, алгоритмы являются ресурсом, который исполняет моделируемое действие [52].

Следующий уровень декомпозиции функциональной модели демонстрирует функционирование системы обнаружения вредоносных программ в ОС для мобильных устройств в целом. Функциональная модель, отображающая принцип ее работы, представлена на рисунке 2.2.

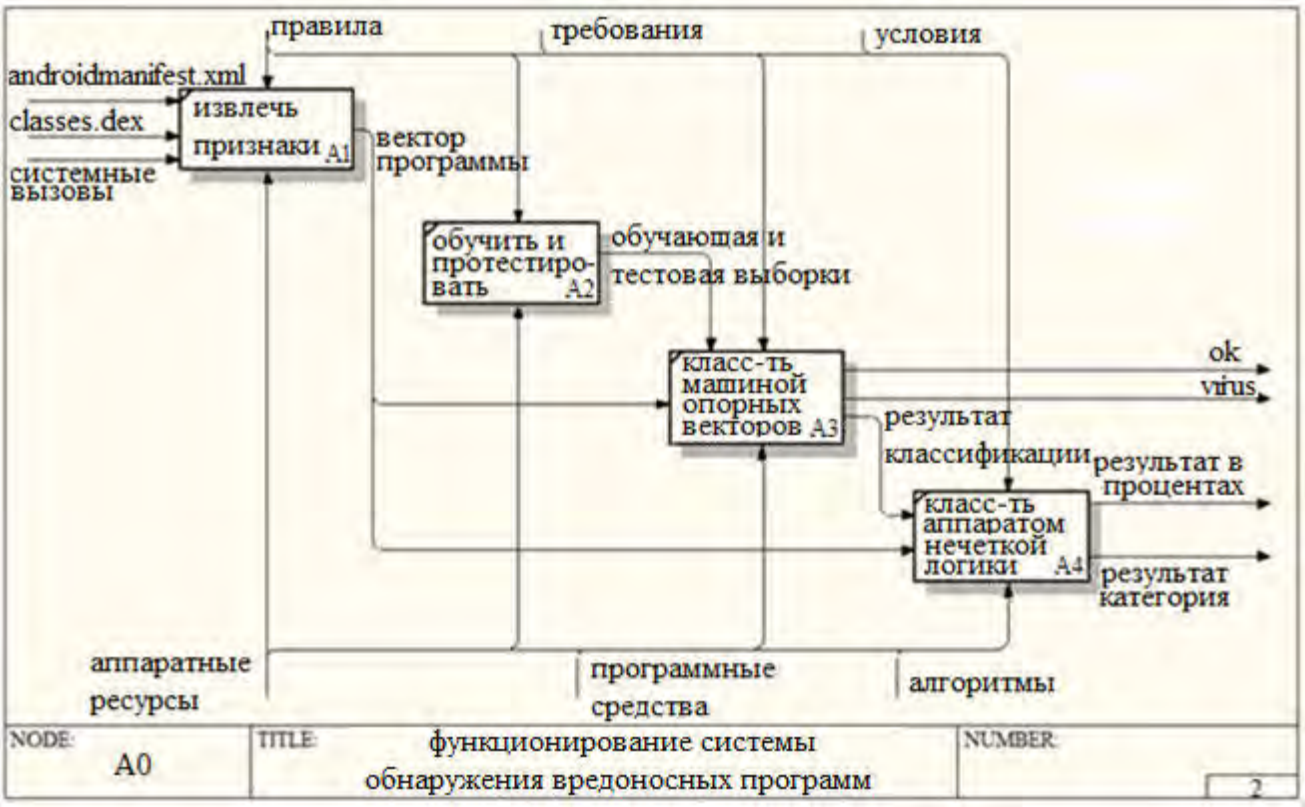


Рисунок 2.2 – Функциональная модель A1 работы системы обнаружения вредоносных программ в ОС Android

В блоке А1 “извлекать признаки” выполняется анализ поступающих на вход компонентов программы (файлов androidmanifest.xml, classes.dex и системных вызовов) и формируется вектор данной программы. Данный вектор подается на блоки А3 “машина опорных векторов” и А4 “нечеткая логика”, в которых выполняется его классификация в один из двух классов: virus, ok, а также дополнительная классификация с учетом условий и результата работы машины опорных векторов.

Следующий уровень декомпозиции функциональной модели, представленный на рисунке 2.3, демонстрирует работу функционального блока А1 “извлечь признаки”.

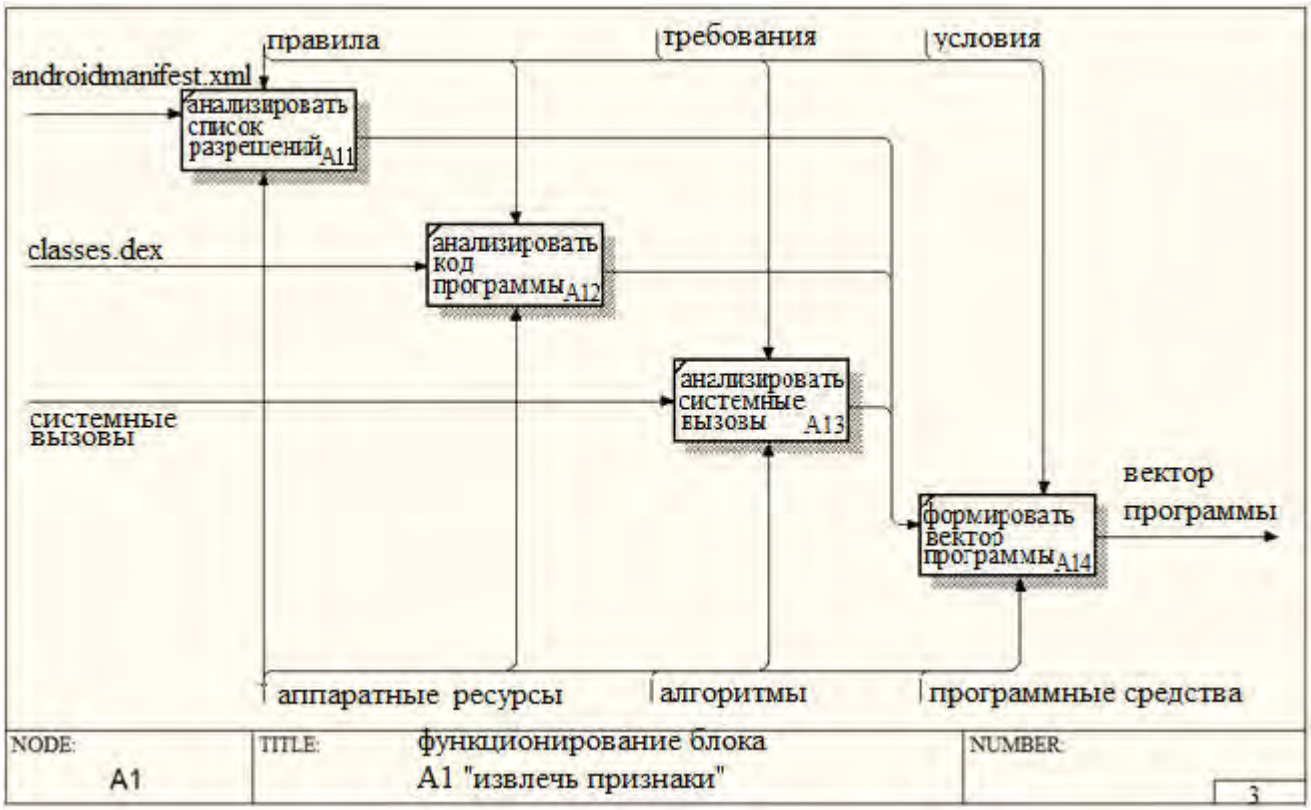


Рисунок 2.3 – Функциональная модель блока А1 “извлекать признаки”

В блоке “извлечь признаки” формируется вектор программы, который характеризует поведение самой программы по наличию в ней разрешений, подозрительного кода, времени обращения к системным вызовам. Результатом работы данного блока является вектор программы, который будет в дальнейшем классифицирован машиной опорных векторов и нечеткой логикой.

Таким образом, выполняется извлечение признаков и преобразование данных признаков в вектор рассматриваемой программы. Далее он классифицируется с отнесением к одному из двух классов и выдает результат в виде процентов с присвоением категории опасности программы в ОС для мобильных устройств (на примере ОС Android). Функциональная модель IDEF0 позволила отобразить структуру и функции системы обнаружения вредоносных программ, а также объекты и потоки информации, связывающие эти функции.

Чтобы визуально оценить и рассмотреть процесс работы системы обнаружения вредоносных программ, необходимо построить информационную модель, отображающую структуру и содержание информационных потоков, необходимых для поддержки функций данной системы. Для этого построим информационную модель IDEF1X, раскрывающую информационное содержание работы системы обнаружения вредоносных программ. Информационная модель представляет собой логическую структуру, содержащую информацию об объекте исследования, которая позволяет отследить процесс работы системы обнаружения вредоносных программ в мобильной ОС. На рисунке 2.4 представлена построенная информационная модель работы системы обнаружения вредоносных программ в мобильной ОС. Блок “androidmanifes.xml” содержит в себе перечень разрешений, запрашиваемых прикладной программой, блок “classes.dex” содержит исполняемый код прикладной программы, блок “системные вызовы” содержит информацию о времени обращения с системными вызовами. Происходит сбор информации из описанных выше блоков для того, чтобы описать программу в виде вектора. Данное действие выполняет блок с названием “извлечение признаков”.

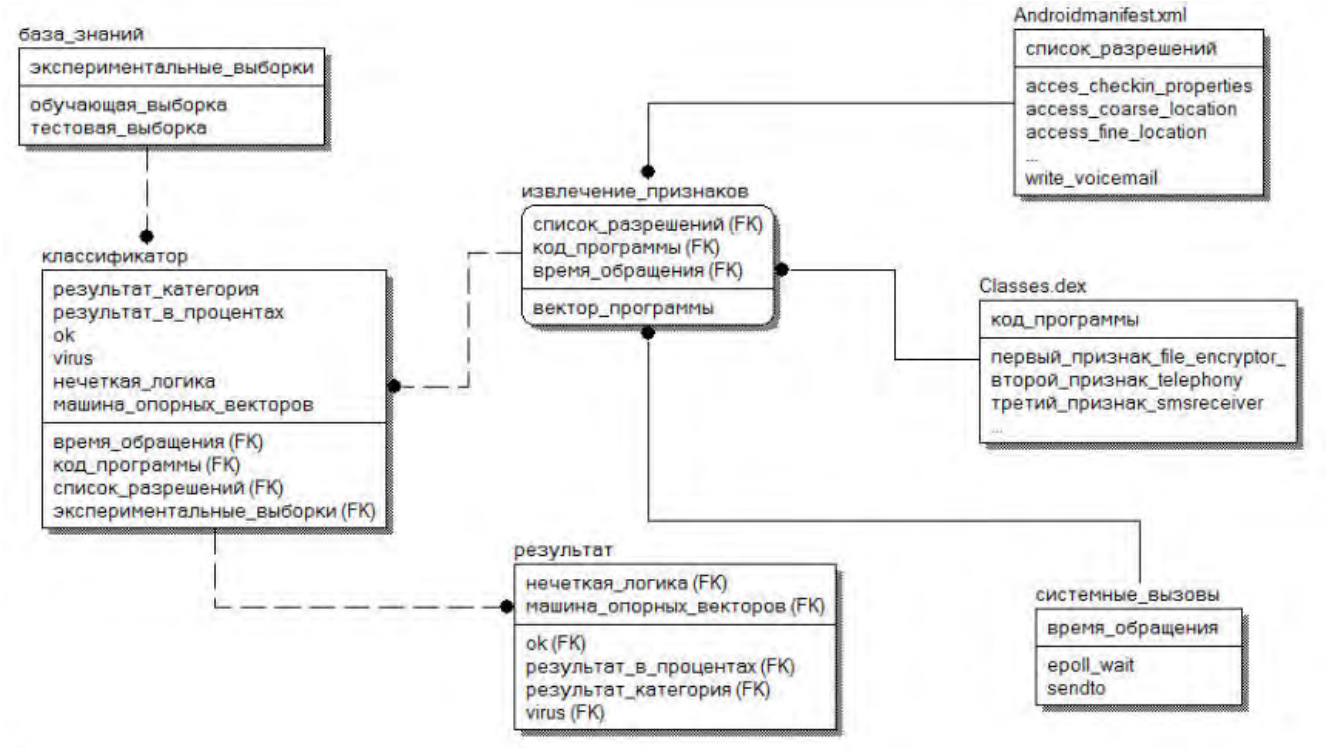


Рисунок 2.4 – Информационная модель процесса функционирования системы обнаружения вредоносных программ в ОС для мобильных устройств

Блок “база знаний” содержит в себе экспериментальные данные, представленные в виде выборки. На данной выборке выполняется обучение и тестирование машины опорных векторов. Обученной и протестированной машине опорных векторов подается вектор описанной программы и ею выполняется классификация на два класса: virus и ok, а также применяется нечеткая логика, которая позволяет выполнить уточнение результата с учетом сильных помех в векторе. Нечеткая логика показывает результат в процентах и категории опасности рассматриваемой программы.

В процессе функционирования мобильное устройство может взаимодействовать с облачными сервисами (облачные вычисления), средой виртуализации и прочими современными сервисами, так как в нем реализован широкий спектр функций и множество современных услуг. Облачный сервис выступает в качестве источника распространения прикладных программ, хранения информации и т.д. На рисунке 2.5 изображена многоуровневая

иерархическая система, реализующая интеллектуальную составляющую обнаружения вредоносных программ.

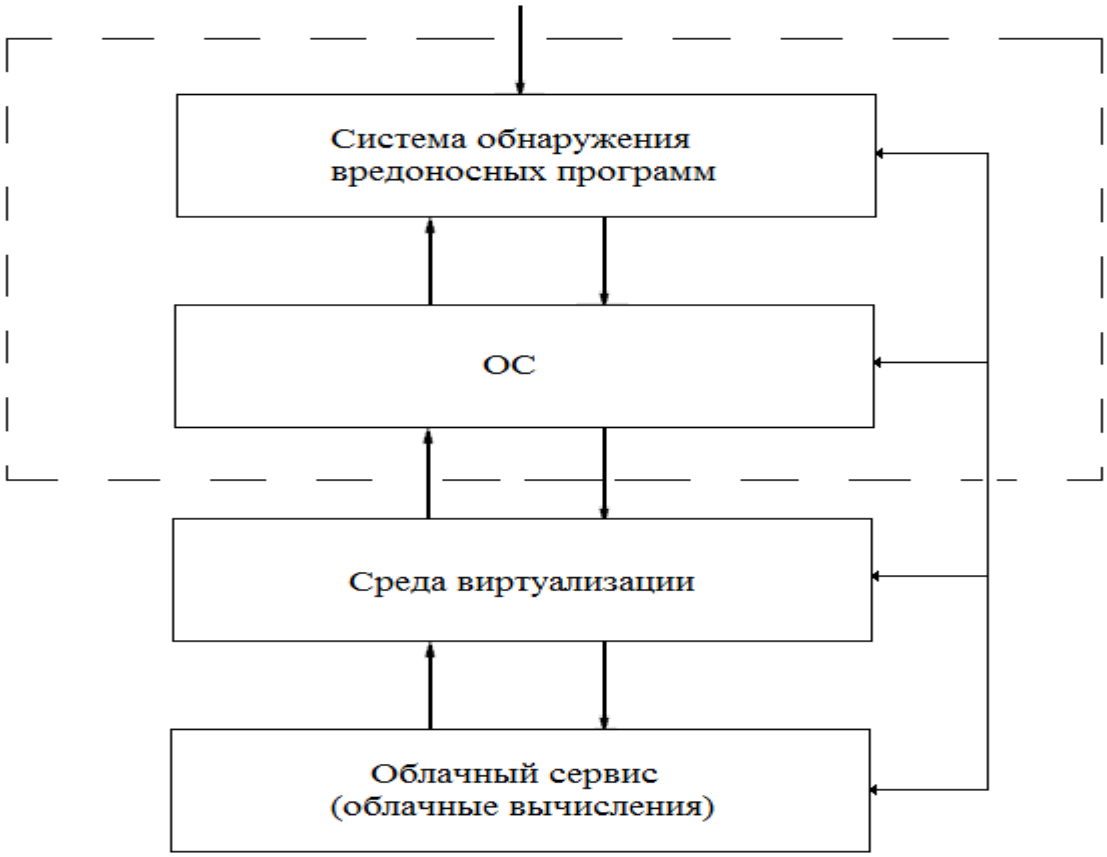


Рисунок 2.5 – Ориентированность работы системы обнаружения вредоносных программ

В рамках данной работы будет разработана методика, которая нацелена на обнаружение вредоносных программ в ОС для мобильных устройств. Она является универсальной и может быть полезной как на других уровнях относительно виртуальной машины и облачного сервиса, так и в альтернативных ОС для мобильных устройств.

Таким образом, построенная функциональная модель IDEF0 позволила отобразить структуру и функции системы обнаружения вредоносных программ, а также объекты и потоки информации, связывающие эти функции. Информационная модель IDEF1X позволила отобразить структуру и содержание

информационных потоков, необходимых для выполнения функций системы обнаружения вредоносных программ.

2.2 Исследование множества вредоносных программ

В настоящее время вредоносные программы в ОС для мобильных устройств (на примере Android) находятся в свободном доступе и их можно найти в сети Интернет [109]. Анализ вредоносных программ позволил выявить свойства и особенности, присущие поведению вредоносных программ, понять принцип работы и сформировать экспериментальную выборку для разработки будущей системы обнаружения вредоносных программ. В таблице 2.1 представлен перечень рассматриваемого множества вредоносных программ.

Таблица 2.1 – Рассматриваемые вредоносные программы

Название	Описание
Android/SpySMS Trojan-Spy.AndroidOS.Zbot.a Android.Smssniffer AndroidSpySMS AndroidOS_SMSREP.B	Перехватывает смс-сообщения и отправляет на сервер запрос, в котором передает номер “зараженного” мобильного устройства, содержимое смс и идентификатор “зараженного” устройства. Такое поведение свойственно банковской троянской программе, нацеленной на двухфакторную авторизацию
DroidKungFu	Дает возможность получить доступ к мобильному устройству с правами привилегированного пользователя для сбора информации об устройстве и пользователе
FlashPlayer	Отправляет смс-сообщения на короткие номера
SimpleLocker	Взаимодействует с сервером и шифрует файлы
Vanilla Music Player	Отправляет смс-сообщения на короткие номера

Анализ вредоносных программ проводился в три этапа:

1. Анализ файла манифест и ресурсов программы, вследствие того, что обзор структуры в перовой главе показал важность данного файла в ОС для мобильных устройств.

2. Декомпиляция программы с целью привести код в “читаемый” вид, так как код программы находится в скомпилированном виде.

3. Анализ кода и файла манифест программ на наличие “подозрительных” строк, не соответствующих функционалу программы.

Любая программа представляет собой файл apk – архивный файл типа zip. В файле apk находятся стандартные файлы и папки: meta-inf, res, assets, androidmanifest.xml, resources.arsc, classes.dex. meta-inf включает в себя информацию о файле apk: контрольные суммы, сертификаты. Res, assets, resources.arsc содержат мультимедиафайлы, которые необходимы для функционирования программы. Файл манифест включает в себя информацию о программе, а именно: аппаратные и программные требования, необходимые для функционирования программы, список разрешений, запрашиваемый программой, название задействованных классов и библиотек. classes.dex содержит программный код (см. главу 1). Данный файл был декомпилирован и открыт так называемой программой apktools, которая позволяет привести код анализируемой программы в читаемый вид.

Ключевую роль играет файл androidmanifest.xml, содержащий список прав на запуск того или иного сервиса для управления им. Система разрешений в ОС для мобильных устройств (на примере Android) включает в себя 152 разрешения. Каждое разрешение отвечает за доступ к тому или иному аппаратному или программному ресурсу. Проанализировав список, можно сделать вывод о критичности запускаемого сервиса. В приложении А описаны все 152 разрешения. Путем анализа критичности и значимости каждому разрешению присвоено значение 0 – приемлемо (безопасно), 1 – опасно, то есть при предоставлении доступа на данное разрешение могут быть задействованы

программно-аппаратные ресурсы, которые вследствие управления ими вредоносной программой могут нанести ущерб.

Описание свидетельствует о том, что определённое число разрешений являются критичными, то есть, дав права программе на работу, мобильное устройство будет выполнять все функции, присущие вредоносным программам.

Вследствие детального анализа в исходном коде перечня рассматриваемых вредоносных программ, а также дополнительной информации, полученной из сети Интернет, были выявлены признаки, характеризующие их поведение:

- наличие файла формата .txt, содержащего списки номеров (например, smsc.txt (7921, 7923, 7924 и т.д.)), в файлах вредоносных программ, который находится в скрытом виде, он содержит в себе координаты: адреса и номера отправки платных смс-сообщений или иных запросов;
- наличие обфускации кода. Код программы может быть зашифрован или запутан с целью скрытия кода, для того чтобы исключить возможность его прочтения и анализа;
- класс, отвечающий за шифрование file encryptor (contex). Класс задействует методы, которые отвечают за шифрование данных;
- наличие класса telephony – отвечает за действия с телефоном;
- наличие класса smsreceiver – отвечает за работу с смс-сообщениями;
- наличие класса smsblockerthead;
- наличие субкласса smsblockerthead;
- служба org.torproject.android.service.tor_service, позволяющая устанавливать анонимное сетевое соединение, защищённое от прослушивания;
- наличие файла в базе сигнатур мобильной антивирусной программы. Классический метод сигнатурного анализа работает эффективно. В связке с динамическим анализом эффективность и качество обнаружения возрастет;
- система разрешений на использование сервисов ОС для мобильных устройств (на примере Android). При правильном подходе и анализе системы

разрешений можно предотвратить возможность инсталляции вредоносной программы в ОС для мобильных устройств на начальном этапе.

Обобщив и проанализировав всю собранную информацию, можно составить модель поведения вредоносных программ на основе выявленных признаков. В результате проделанной работы была сформирована экспериментальная выборка с целью анализа поведения для обнаружения вредоносных программ ОС для мобильных устройств.

2.3 Описание работы модели ОС для мобильных устройств с помощью скрытой марковской модели с учетом воздействия вредоносных программ

Одной из основных моделей случайных процессов, используемой в прогнозировании, является скрытая марковская модель. Такими моделями, которые могут быть включены в системы поддержки принятия решений, описывается большое количество процессов. В данном случае выполняется описание процесса работы мобильной ОС с учетом воздействия вредоносных программ – время нахождение системы в рабочем и нерабочем состояниях [43].

Скрытая марковская модель позволит перейти от качественных оценок процесса работы ОС для мобильных устройств к количественным оценкам – количества времени пребывания в одном из состояний.

Пусть рассматривается система S – ОС для мобильных устройств, имеющая n возможных состояний S1, S2, S3, …, Sn.

Имеется ряд дискретных состояний системы S, в которых она может находиться: S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13.

Опишем все возможные состояния ОС для мобильных устройств:

S1 – стабильное рабочее состояние ОС для мобильных устройств;

S2 – загрузка прикладных программ из известного или неизвестного источника;

S3 – прикладные программы выполняют запрос разрешений к пользователю согласно заложенному в них функционалу;

S4 – инсталлируется код полученной прикладной программы;

S5 – вредоносная программа находится в состояние покоя (ожидания);

S6 – вредоносная программа выполняет сбор информации о пользователе и устройстве;

S7 – сторонние средства защиты (антивирусная программа);

S8 – база сигнатур антивирусной программы;

S9 – встроенные средства защиты (песочница, система разрешений (глава 1));

S10 – ремонт, восстановление ОС для мобильных устройств;

S11 – отказ, нерабочее состояние мобильного устройства;

S12 – вредоносная программа активируется, то есть выполняет “захват” управления над устройством;

S13 – вредоносная программа выполняет заложенный в нее функционал: пересылает смс, получает контроль над устройством и выполняет прочие вредоносные действия.

Скрытые марковские модели с дискретными состояниями удобно изображать в виде графа состояний. Кружками обозначены состояния S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13 ОС для мобильных устройств S, а стрелки показывают возможные переходы из состояния в состояние. Задержки в прежнем состоянии изображаются петлей – стрелкой, направленной из данного состояние в него же. На рисунке 2.6 представлена скрытая марковская модель процесса функционирования мобильной ОС, составленная с учетом деятельности вредоносных программ, а также имеющихся в настоящее время встроенных и прикладных средств защиты.

Все интенсивности потоков событий, которые переводят ОС для мобильных устройств из одного состояния в другое состояние, постоянные, то есть простейшие потоки $\lambda_{ij} = const$.

Поскольку время перехода мобильной ОС из состояния в состояние не-фиксированное, будем считать, что $t \to \infty$. В таком случае $p_n(t)$ будут стремиться к пределам, которые называются предельными вероятностями состояний:

$$\lim_{t\to\infty} p_i(t) = p_i \ (i = 1, 2, \dots, n).$$

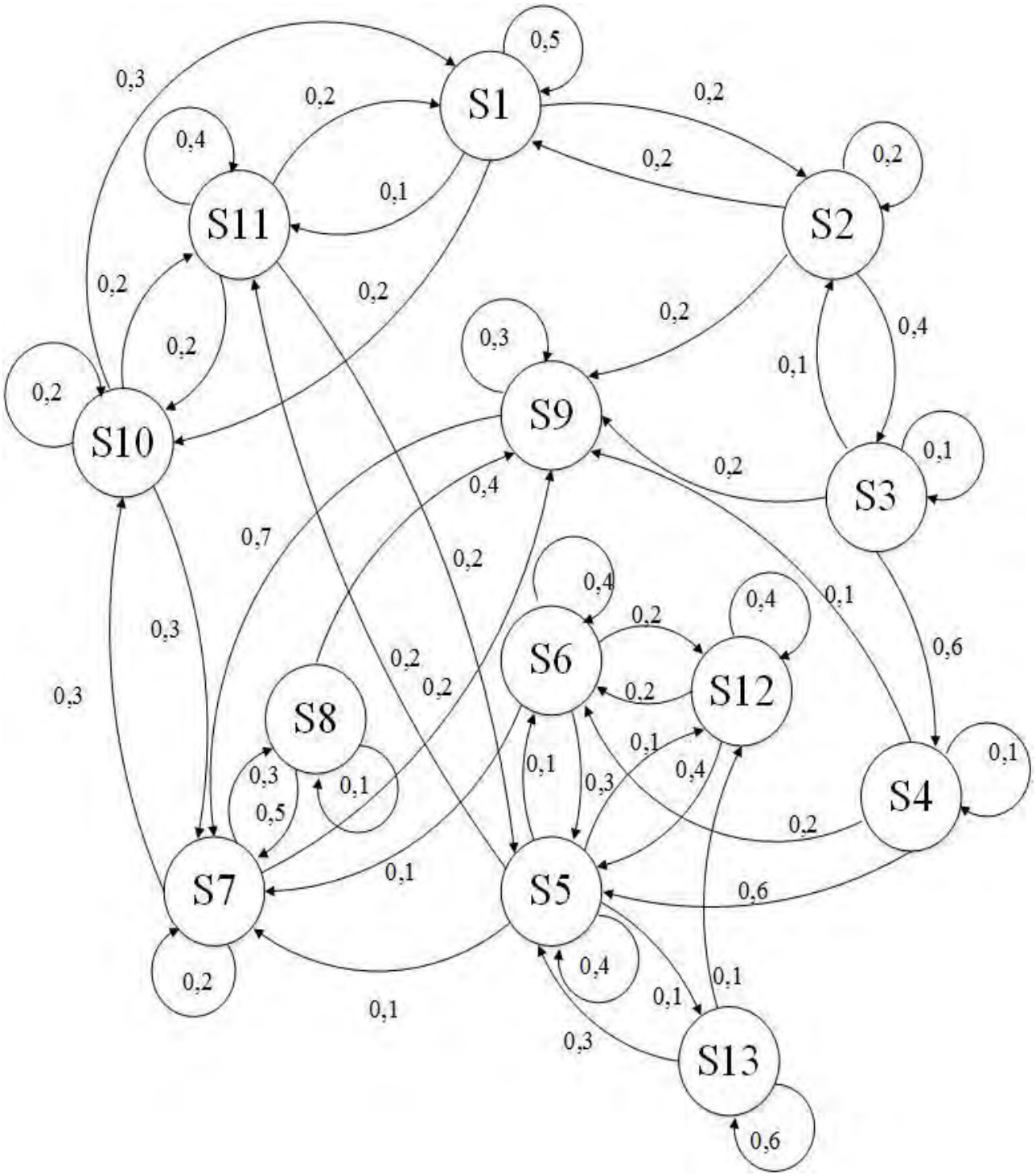


Рисунок 2.6 – Граф состояний ОС для мобильных устройств

Вероятностью перехода (переходной вероятностью) на -м шаге из состояния S_i в состояние S_j называется условная вероятность того, что система S после k-го шага окажется в состоянии S_j при условии, что непосредственно перед этим (после k-1 шага) она находилась в состоянии S_i [65]. Поскольку система может пребывать в одном из n состоянии, то для каждого момента времени t необходимо

задать n^2 вероятностей перехода P_{ij}, которые удобно представить в виде матрицы. Составим матрицу условных переходных вероятностей для полученной скрытой марковской модели. Матрица условных переходных вероятностей представлена в виде таблицы 2.2.

Таблица 2.2 – Матрица условных переходных вероятностей

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
S1	$S_{1.1}$	$S_{1.2}$	0	0	0	0	0	0	0	$S_{1.10}$	$S_{1.11}$	0	0
S2	$S_{2.1}$	$S_{2.2}$	$S_{2.3}$	0	0	0	0	0	$S_{2.9}$	0	0	0	0
S3	0	$S_{3.2}$	$S_{3.3}$	$S_{3.4}$	0	0	0	0	$S_{3.9}$	0	0	0	0
S4	0	0	0	$S_{4.4}$	$S_{4.5}$	$S_{4.6}$	0	0	$S_{4.9}$	0	0	0	0
S5	0	0	0	0	$S_{5.5}$	$S_{5.6}$	$S_{5.7}$	0	0	0	$S_{5.11}$	$S_{5.12}$	$S_{5.13}$
S6	0	0	0	0	$S_{6.5}$	$S_{6.6}$	$S_{6.7}$	0	0	0	0	$S_{6.12}$	0
S7	0	0	0	0	0	0	$S_{7.7}$	$S_{7.8}$	$S_{7.9}$	$S_{7.10}$	0	0	0
S8	0	0	0	0	0	0	$S_{8.7}$	$S_{8.8}$	$S_{8.9}$	0	0	0	0
S9	0	0	0	0	0	0	0	$S_{9.7}$	$S_{9.9}$	0	0	0	0
S10	$S_{10.1}$	0	0	0	0	0	$S_{10.7}$	0	0	$S_{10.10}$	$S_{10.11}$	0	0
S11	$S_{11.1}$	0	0	0	$S_{11.5}$	0	0	0	0	$S_{11.10}$	$S_{11.11}$	0	0
S12	0	0	0	0	$S_{12.5}$	$S_{12.6}$	0	0	0	0	0	$S_{12.12}$	0
S13	0	0	0	0	$S_{13.5}$	0	0	0	0	0	0	$S_{13.12}$	$S_{13.13}$

Значения переходных вероятностей S_{ij} были определены экспертным методом, в качестве экспертов выступали разработчики ОС для мобильных устройств, администраторы информационной безопасности компании Яндекс, занимающиеся внедрением математических методов для более эффективного решения различных задач, возникающих перед службой информационной безопасности, в частности разработками в сфере ОС для мобильных устройств [97]. Данные значения были получены путем группового анкетирования экспертов, а также сбора и анализа информации, полученной из сети Интернет

(таблица 2.3). В приложении Г представлена анкета, согласно которой были получены экспертные оценки.

Скрытые марковские модели описывают сильно связанным графом. Это означает, что в такой системе возможен переход из любого состояния S_i в любое состояние S_j (i,j=1,…,n) за конечное число шагов.

Таблица 2.3 – Матрица условных переходных вероятностей, выраженная экспертным методом

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
S1	0.5	0.2	0	0	0	0	0	0	0	0.2	0.1	0	0
S2	0.2	0.2	0.4	0	0	0	0	0	0.2	0	0	0	0
S3	0	0.1	0.1	0.6	0	0	0	0	0.2	0	0	0	0
S4	0	0	0	0.1	0.6	0.2	0	0	0.1	0	0	0	0
S5	0	0	0	0	0.4	0.1	0.1	0	0	0	0.2	0.1	0.1
S6	0	0	0	0	0.3	0.4	0.1	0	0	0	0	0.2	0
S7	0	0	0	0	0	0	0.2	0.3	0.2	0.3	0	0	0
S8	0	0	0	0	0	0	0.5	0.1	0.4	0	0	0	0
S9	0	0	0	0	0	0	0	0.7	0.3	0	0	0	0
S10	0.3	0	0	0	0	0	0.3	0	0	0.2	0.2	0	0
S11	0.2	0	0	0	0.2	0	0	0	0	0.2	0.4	0	0
S12	0	0	0	0	0.4	0.2	0	0	0	0	0	0.4	0
S13	0	0	0	0	0.3	0	0	0	0	0	0	0.1	0.6

Для скрытой марковской модели при достаточно большом времени функционирования при $t \to \infty$ наступает стационарный режим, при котором вероятности P_i состояний системы не зависят от времени и не зависят от распределения вероятностей в начальный момент времени, т.е. P_i =const.

Каждая компонента P_i вектора таких стационарных вероятностей характеризует среднюю долю времени, в течение которого система находится в рассматриваемом состоянии S_i за время наблюдения, измеряемое k шагами.

Для определения стационарных вероятностей P_i нахождения системы в состоянии S_i $(i = 1, \dots, n)$ нужно составить систему n линейных однородных алгебраических уравнений с n неизвестными:

$$P_i = \sum_{j=1}^{n} P_j P_{ji}, (i = 1, \dots, n).$$

Причем искомые вероятности должны удовлетворять нормировочному условию

$$\sum_{i=1}^{n} P_i = 1.$$

Таким образом, при $t \to \infty$ в ОС S для мобильных устройств устанавливается предельный стационарный режим, то есть она случайным образом изменяет состояния, но вероятность каждого состояния не зависит от времени: каждые состояния выполняются с определенной постоянной вероятностью. Данная вероятность представляет собой среднее относительное время пребывания системы в каком-либо определенном состоянии.

Систему линейных алгебраических уравнений удобно составлять по размеченному графу состояний. При этом в левой части уравнения записывается вероятность состояний, соответствующих рассматриваемой вершине графа, а в правой части – сумма произведений. Число слагаемых соответствует числу дуг графа, входящих в рассматриваемое состояние. Каждое слагаемое представляет произведение вероятности того состояния, из которого выходит дуга графа, на переходную вероятность, которой помечена соответствующая дуга графа.

На основании матрицы переходных состояний и графа состояний, а также с помощью нормировочного условия составим для стационарного режима систему линейных алгебраических уравнений и решим их методом Крамера.

$$\begin{cases} \boldsymbol{p}_1 = 0.5\boldsymbol{p}_1 + 0.2\boldsymbol{p}_2 + 0.3\boldsymbol{p}_{10} + 0.2\boldsymbol{p}_{11} \\ \boldsymbol{p}_2 = 0.2\boldsymbol{p}_1 + 0.2\boldsymbol{p}_2 + 0.1\boldsymbol{p}_3 \\ \boldsymbol{p}_3 = 0.4\boldsymbol{p}_2 + 0.1\boldsymbol{p}_3 \\ \boldsymbol{p}_4 = 0.6\boldsymbol{p}_3 + 0.1\boldsymbol{p}_4 \\ \boldsymbol{p}_5 = 0.6\boldsymbol{p}_4 + 0.4\boldsymbol{p}_5 + 0.3\boldsymbol{p}_6 + 0.2\boldsymbol{p}_{11} + 0.4\boldsymbol{p}_{12} + 0.3\boldsymbol{p}_{13} \\ \boldsymbol{p}_6 = 0.2\boldsymbol{p}_4 + 0.1\boldsymbol{p}_5 + 0.4\boldsymbol{p}_6 + 0.2\boldsymbol{p}_{12} \\ \boldsymbol{p}_7 = 0.1\boldsymbol{p}_5 + 0.1\boldsymbol{p}_6 + 0.2\boldsymbol{p}_7 + 0.5\boldsymbol{p}_8 + 0.3\boldsymbol{p}_{10} \\ \boldsymbol{p}_8 = 0.3\boldsymbol{p}_7 + 0.1\boldsymbol{p}_8 + 0.7\boldsymbol{p}_9 \\ \boldsymbol{p}_9 = 0.2\boldsymbol{p}_2 + 0.3\boldsymbol{p}_3 + 0.1\boldsymbol{p}_4 + 0.2\boldsymbol{p}_7 + 0.4\boldsymbol{p}_8 + 0.3\boldsymbol{p}_9 \\ \boldsymbol{p}_{10} = 0.2\boldsymbol{p}_1 + 0.3\boldsymbol{p}_7 + 0.2\boldsymbol{p}_{10} + 0.4\boldsymbol{p}_{11} \\ \boldsymbol{p}_{11} = 0.1\boldsymbol{p}_1 + 0.3\boldsymbol{p}_5 + 0.2\boldsymbol{p}_{10} + 0.4\boldsymbol{p}_{11} \\ \boldsymbol{p}_{12} = 0.1\boldsymbol{p}_5 + 0.2\boldsymbol{p}_6 + 0.4\boldsymbol{p}_{12} + 0.1\boldsymbol{p}_{13} \\ \boldsymbol{p}_{13} = 0.1\boldsymbol{p}_5 + 0.6\boldsymbol{p}_{13} \end{cases}$$

Путем простых математических преобразований упростим систему алгебраических уравнений, в результате чего в левой части останутся нули и единицы, а в правой – неизвестные. Решим полученные уравнения методом Крамера.

$$\begin{cases} 0 = -0.5\boldsymbol{p}_1 + 0.2\boldsymbol{p}_2 + 0.3\boldsymbol{p}_{10} + 0.2\boldsymbol{p}_{11} \\ 0 = 0.2\boldsymbol{p}_1 - 0.8\boldsymbol{p}_2 + 0.1\boldsymbol{p}_3 \\ 0 = 0.4\boldsymbol{p}_2 - 0.9\boldsymbol{p}_3 \\ 0 = 0.6\boldsymbol{p}_3 - 0.9\boldsymbol{p}_4 \\ 0 = 0.6\boldsymbol{p}_4 - 0.6\boldsymbol{p}_5 + 0.3\boldsymbol{p}_6 + 0.2\boldsymbol{p}_{11} + 0.4\boldsymbol{p}_{12} + 0.3\boldsymbol{p}_{13} \\ 0 = 0.2\boldsymbol{p}_4 + 0.1\boldsymbol{p}_5 - 0.6\boldsymbol{p}_6 + 0.2\boldsymbol{p}_{12} \\ 0 = 0.1\boldsymbol{p}_5 + 0.1\boldsymbol{p}_6 - 0.8\boldsymbol{p}_7 + 0.5\boldsymbol{p}_8 + 0.3\boldsymbol{p}_{10} \\ 0 = 0.3\boldsymbol{p}_7 - 0.9\boldsymbol{p}_8 + 0.7\boldsymbol{p}_9 \\ 0 = 0.2\boldsymbol{p}_2 + 0.3\boldsymbol{p}_3 + 0.1\boldsymbol{p}_4 + 0.2\boldsymbol{p}_7 + 0.4\boldsymbol{p}_8 - 0.7\boldsymbol{p}_9 \\ 0 = 0.2\boldsymbol{p}_1 + 0.3\boldsymbol{p}_7 - 0.8\boldsymbol{p}_{10} + 0.4\boldsymbol{p}_{11} \\ 0 = 0.1\boldsymbol{p}_1 + 0.3\boldsymbol{p}_5 + 0.2\boldsymbol{p}_{10} - 0.6\boldsymbol{p}_{11} \\ 0 = 0.1\boldsymbol{p}_5 + 0.2\boldsymbol{p}_6 - 0.6\boldsymbol{p}_{12} + 0.1\boldsymbol{p}_{13} \\ 1 = \boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3 + \boldsymbol{p}_4 + \boldsymbol{p}_5 + \boldsymbol{p}_6 + \boldsymbol{p}_7 + \boldsymbol{p}_8 + \boldsymbol{p}_9 + \boldsymbol{p}_{10} + \boldsymbol{p}_{11} + \boldsymbol{p}_{12} + \boldsymbol{p}_{13} \end{cases}$$

Из полученного уравнения составим матрицу переходных состояний. Матрица переходных состояний системы представлена в виде таблицы 2.4.

Таблица 2.4 – Матрица переходных состояний

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	b
1	-0.5	0.2	0	0	0	0	0	0	0	0.3	0.2	0	0	0
2	0.2	-0.8	0.1	0	0	0	0	0	0	0	0	0	0	0
3	0	0.4	-0.9	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.6	-0.9	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0.6	-0.6	0.3	0	0	0	0	0.2	0.4	0.3	0
6	0	0	0	0.2	0.1	-0.6	0	0	0	0	0	0.2	0	0
7	0	0	0	0	0.1	0.1	-0.8	0.5	0	0.3	0	0	0	0
8	0	0	0	0	0	0	0.3	-0.9	0.7	0	0	0	0	0
9	0	0.2	0.2	0.1	0	0	0.2	0.4	-0.7	0	0	0	0	0
10	0.2	0	0	0	0	0	0.3	0	0	-0.8	0.2	0	0	0
11	0.1	0	0	0	0.2	0	0	0	0	0.2	-0.6	0	0	0
12	0	0	0	0	0.1	0.2	0	0	0	0	0	-0.6	0.1	0
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Результаты расчета матрицы представлены в таблице 2.5.

Таблица 2.5 – Результаты расчета полученной матрицы

Дискриминант	Расчет предельной вероятности
1	2
D = 0.0045	P1 = D1 / D = 0.0005 / 0.0045 = 0.11
D1 = 0.0005	P2 = D2 / D = 0.0001 / 0.0045 = 0.03
D2 = 0.0001	P3 = D3 / D = 0.00006 / 0.0045 = 0.013
D3 = 0.00006	P4 = D4 / D = 0.00004 / 0.0045 = 0.008
D4 = 0.00004	P5 = D5 / D = 0.0003 / 0.0045 = 0.067
D5 = 0.0003	P6 = D6 / D = 0.00009 / 0.0045 = 0.02
D6 = 0.00009	P7 = D7 / D = 0.0008 / 0.0045 = 0.17

Окончание таблицы 2.5

1	2
D8 = 0.0008	P9 = D9 / D = 0.0008 / 0.0045 = 0.17
D9 = 0.0008	P10 = D10 / D = 0.0005 / 0.0045 = 0.11
D10 = 0.0005	P11 = D11 / D = 0.0003 / 0.0045 = 0.08
D11 = 0.0003	P12 = D12 / D = 0.00009 / 0.0045 = 0.02
D12 = 0.00009	P13 = D13 / D = 0.00008 / 0.00453 = 0.016
D13 = 0.00007	P1 = D1 / D = 0.0005 / 0.0045 = 0.11

Таким образом, рассчитаны предельные вероятности времени пребывания системы в одном из состояний, они равны: P1 = 0.11, P2 = 0.03, P3 = 0.013, P4 = 0.008, P5 = 0.07, P6 = 0.02, P7 = 0.17, P8 = 0.19, P9 = 0.17, P10 = 0.11, P11 = = 0.08, P12 = 0.02, P13 = 0.017.

Каждая компонента P_i вектора стационарных вероятностей характеризует среднюю долю времени, в течение которого ОС для мобильных устройств находится в рассматриваемом состоянии S_i. Можно посчитать, сколько в общем наша система находится в рабочем и нерабочем состояниях. Анализируя каждое состояние, выделим из множества состояний те, которые относятся к рабочему состоянию ОС для мобильных устройств: $S_{\text{р}} =$ S1, S2, S3, S4, S7, S8, S9.

$$(1 - S_{\text{н}}) \times 100\ \% = S_{\text{р}},$$

где $S_{\text{н}}$ – множество нерабочих состояний ОС для мобильных устройств,

$S_{\text{р}}$ – множество рабочих состояний ОС для мобильных устройств.

Состояния $S_{\text{н}} =$ S5, S6, S10, S11, S12, S13 относятся к нерабочему состоянию.

$$\left(1 - S_{\text{р}}\right) \times 100\ \% = S_{\text{н}}.$$

Таким образом, можно сделать вывод, что система находится в рабочем состоянии 68,7 % времени, остальное время находится под воздействием вредоносных программ. Следовательно, достаточно большой процент времени система находится в нерабочем состоянии, что свидетельствует о том, что 31,3 %

ресурсов ОС для мобильных устройств расходует на пребывание в нерабочем состоянии.

2.4 Разработка нечеткой когнитивной карты и оценка риска информационной безопасности ОС для мобильных устройств с учетом имеющихся в настоящее время средств защиты информации

С помощью когнитивной карты можно выполнять качественные оценки степени влияния одних компонентов данной модели на другие, а нечеткие когнитивные карты помимо этого отражают силу причинно-следственных связей посредством нечетких отношений, которые задаются экспертами: оказывает слабое влияние, среднее и сильное [11, 97].

Для того чтобы построить нечеткую когнитивную карту, необходимо сформировать список компонентов, из которых она будет состоять, установить причинно-следственные связи между данными компонентами и определить их переменные состояния.

С помощью анализа, проведенного в первой главе, и моделирования во второй главе, были выделены основные компоненты и условно разделены на 3 группы:

1. Ресурсы ОС для мобильных устройств, которые необходимо защищать:

- ядро ОС для мобильных устройств. Ядро является центром управления и работает с правами привилегированного пользователя;
- драйвера. Получив доступ к драйверам, можно осуществлять управление работой аппаратной части ОС для мобильных устройств;
- программное обеспечение. Управление программами как системными, так и прикладными в своих целях может нанести огромный ущерб;
- система прав (разрешений). С помощью системы разрешений можно предоставить все возможные права и обеспечить достаточно ресурсов для работы вредоносных программ.

2. Перечень факторов и средств, создающих определенные условия, которые могут повлечь опасность нарушения информационной безопасности ОС для мобильных устройств (см. глава 1):

- эксплойт;
- неофициальная или модернизированная ОС для мобильных устройств;
- человеческий фактор;
- кража;
- вредоносные программы.

3. Виды ущерба от воздействия угроз:

- финансовые потери. В результате воздействия угроз пользователь может понести финансовые потери, например, в результате отправки платных смс, заражения банковских программ и т.д.;
- личная информация. Такая информация, как номера, мультимедиафайлы, деловая переписка, документы и т.д.;
- контроль над устройством. За счет контроля над устройством можно отслеживать, например, действия пользователя, расположение устройства и т.д.

Для того чтобы установить силу связей между компонентами, использовались нечеткие отношения на шкале от 0 до 1. Нечеткие отношения задавались экспертами с учетом имеющихся на сегодняшнее время средств защиты информации в ОС для мобильных устройств:

- 0 – не влияет – 0 (0);
- 0.1 – слабое влияние – Low (L);
- 0.3 – среднее влияние – Medium (M);
- 1 – сильное влияние – High (H).

Построим нечеткую когнитивную карту для оценки риска до внедрения интеллектуальной системы обнаружения вредоносных программ.

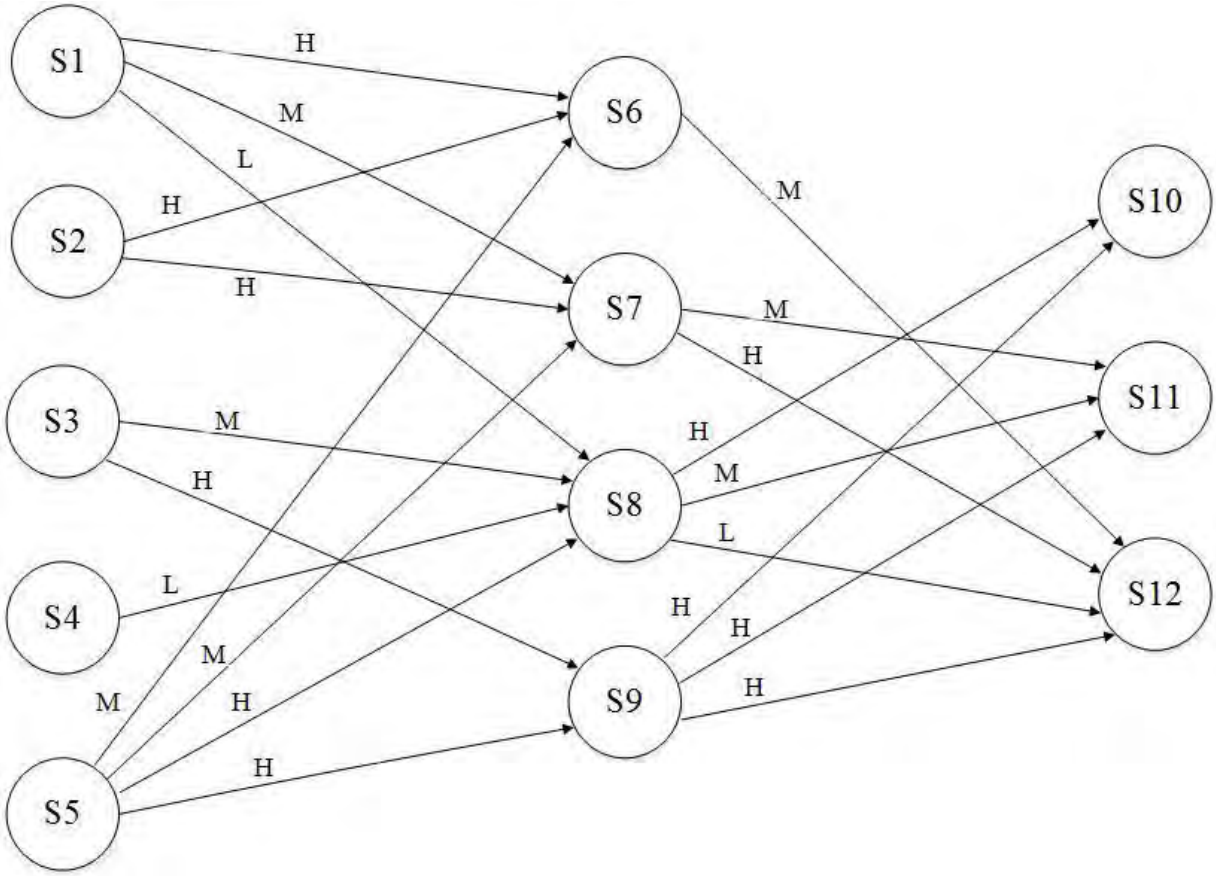


Рисунок 2.7 – Нечеткая когнитивная карта до внедрения интеллектуальной системы обнаружения вредоносных программ

Обозначения и описание компонентов представлены в таблице 2.6.

Таблица 2.6 – Название и описание компонентов нечеткой когнитивной карты

Обозначение компонента	Название компонента
1	2
S1	Эксплоит
S2	Неофициальная или модернизированная ОС для мобильных устройств, которая может содержать в себе сторонние программы
S3	Человеческий фактор

Окончание таблицы 2.6

1	2
S4	Кража мобильного устройства
S5	Вредоносные программы
S6	Ядро ОС для мобильных устройств
S7	Драйвера
S8	Программное обеспечение
S9	Система прав (разрешений)
S10	Финансовые потери
S11	Личная информация (документы, мультимедиа файлы, контакты и т.д.)
S12	Контроль над устройством

Нечеткая когнитивная карта представляет собой взвешенный орграф, описывающий состав компонентов и взаимодействие между ними в виде кортежа множеств [39]:

$$\text{НКК} = \{S, M\},$$

где S – множество заданных компонентов (ресурсы ОС для мобильных устройств, угрозы, ущерб, имеющиеся и внедренные средства защиты информации);
M – множество весов связей между компонентами.

С помощью нечеткой когнитивной карты можно оценить влияние как отдельной угрозы, так и совокупности угроз на тот или иной фактор путем вычисления веса соответствующего пути с учетом веса входящих в него связей. Чтобы посчитать вес пути с учетом весов входящих в него связей, воспользуемся правилами нечеткой логики. Для оценки общего эффекта от действия компонента на угрозы на компонент ущерба нужно найти матрицу достижимости [40]:

$$\boldsymbol{F} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}-1} \boldsymbol{M}^{\boldsymbol{i}},$$

где n – число компонентов нечеткой когнитивной карты;

$\boldsymbol{M^i = \|M_{ij}\|_{n\times n}}$ – матрица смежности нечеткой когнитивной карты;

$\boldsymbol{M_{ij}}$ – вес связи между -м и j-м компонентами.

Можно оценить влияние (эффект) произвольного компонента на любой другой компонент. При этом некоторый путь от компонента к компоненту считается непрямым эффектом. Непрямой эффект влияния угроз на ущерб определяется минимальным из значений весов связи, встречающихся на пути $\boldsymbol{F_t}$:

$$F_t(\text{угроза}_i \rightarrow \text{ущерб}_j) = \min\{M_{ij}\}.$$

Полный эффект влияния угроз на ущерб будет определяться:

$$F(\text{угроза}_i \rightarrow \text{ущерб}_j) = \max\{F_1, F_2, F_3, \dots, F_n\},$$

где $\boldsymbol{F_t}$ – непрямой эффект между угрозой и ущербом;

F – полный эффект влияния угрозы на ущерб;

n – число непрямых эффектов (число путей между угрозой и ущербом).

Полный эффект позволяет оценить вероятность реализации угрозы на виды ущерба по формуле:

$$R_{ij} = F(\text{угроза}_i \rightarrow \text{ущерб}_j) \times r_j,$$

где $F(\text{угроза}_i \rightarrow \text{ущерб}_j)$ – полный эффект влияния угроз на ущерб;

$\boldsymbol{r_j}$ – ценность -го ресурса.

Общий риск от действия угроз на все виды ущерба можно посчитать по формуле [18]:

$$\boldsymbol{R = \sum_{ij} v_j R_{ij}},$$

где $\boldsymbol{v_j}$ – значимость (вес) -го ресурса, определяемая экспертно;

$\boldsymbol{R_{ij}}$ – ущерб, нанесенный -й угрозе j-му ресурсу.

Составим матрицу достижимости F для построенной нечеткой когнитивной карты (таблица 2.7).

Таблица 2.7 – Матрица достижимости

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	0	0	0	0	0	H	M	L	0	L	M	M
S2	0	0	0	0	0	H	H	0	0	0	H	H
S3	0	0	0	0	0	0	0	M	H	H	H	H
S4	0	0	0	0	0	0	0	L	0	L	L	L
S5	0	0	0	0	0	M	M	H	H	H	H	H
S6	0	0	0	0	0	0	0	0	0	0	0	M
S7	0	0	0	0	0	0	0	0	0	0	M	H
S8	0	0	0	0	0	0	0	0	0	H	M	L
S9	0	0	0	0	0	0	0	0	0	H	H	H
S10	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	0

Значения переходных вероятностей S_{ij} определили экспертным методом, в качестве экспертов выступали разработчики ОС для мобильных устройств, администраторы информационной безопасности компании Яндекс, занимающиеся внедрением математических методов для более эффективного решения различных задач, возникающих перед службой информационной безопасности, в частности разработками в сфере ОС для мобильных устройств [97]. Данные значения были получены путем группового анкетирования экспертов, а также сбора и анализа информации, полученной из сети Интернет. В приложении Г представлена анкета, согласно которой были получены экспертные оценки.

Таблица 2.8 – Матрица достижимости, построенная с учетом экспертных оценок

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	0	0	0	0	0	1	0.4	0.2	0	0.2	0.4	0.4
S2	0	0	0	0	0	1	1	0	0	0	1	1
S3	0	0	0	0	0	0	0	0.4	1	1	1	1
S4	0	0	0	0	0	0	0	0.2	0	0.2	0.2	0.2
S5	0	0	0	0	0	0.4	0.4	1	1	1	1	1
S6	0	0	0	0	0	0	0	0	0	0	0	0.4
S7	0	0	0	0	0	0	0	0	0	0	0.4	1
S8	0	0	0	0	0	0	0	0	0	1	0.4	0.2
S9	0	0	0	0	0	0	0	0	0	1	1	1
S10	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	0

Можно сделать вывод, что на финансовые потери будут влиять слабо эксплоит, кража, не будет оказывать влияние ОС, а человеческий фактор, вредоносные программы будут влиять сильно.

Общее влияние угроз на финансовые потери составит 2.4, на личную информацию 3.6, на контроль над устройством 3.6. Общий риск будет составлять 9.6.

Для оценки риска необходимо задать ценность каждого целевого фактора, то есть если ценность целевых факторов определить в процентах, то “Финансовые потери” – 30 %, “Личная информация” – 50 %, и “Контроль над устройством” – 20 %. Следовательно, риски от действия угроз на целевые факторы составят: R1-10 = 6; R1-11 = 20; R1-12 = 8; R2-10 = 0; R2-11 = 50; R2-12 = 20; R3-10 = 30; R3-11 = 50; R3-12 = 20; R4-10 = 6; R4-11 = 10; R4-12 = 4; R5-10 = 30; R5-11 = 50; R5-12 = 20.

Таким образом, общий риск с учетом значимости целевых факторов (v1 = 0.39; v2 = 0.2; v3 = 0.41) составит: R = 0.39(6+0+30+6+30) + +0.2(20+50+50+10+50) + 0.41(8+20+20+4+20) = 93.6.

Следовательно, риск влияния угроз (с учетом общего риска) на целевые факторы составляет 93 %, что свидетельствует о необходимости применения дополнительных средств защиты.

2.5 Выводы по второй главе

1. Разработаны функциональная и информационная модели работы системы обнаружения вредоносных программ на основе технологий IDEF0 и IDEF1X, позволяющие в виде схем отобразить структуру и функции системы обнаружения вредоносных программ, структуру и содержание информационных потоков, необходимых для поддержки работы системы обнаружения вредоносных программ.

2. Исследовано множество вредоносных программ на основе анализа выбранного перечня вредоносных программ, что позволило выделить свойства, присущие поведению вредоносных программ, для формирования экспериментальной выборки.

3. Разработана модель работы ОС для мобильных устройств с учетом воздействия вредоносных программ, на основе скрытой марковской модели, что позволило рассчитать время нахождения системы в одном из состояний (рабочем состоянии 68 %, нерабочем 31 % времени), что свидетельствует о необходимости внедрения дополнительных средств защиты.

4. Разработана модель ОС для мобильных устройств с учетом имеющихся в настоящее время средств защиты на основе нечетких когнитивных карт, что позволило рассчитать риск информационной безопасности, до внедрения дополнительной системы защиты информации составляющий 93 %.

ГЛАВА 3 РАЗРАБОТКА МЕТОДИКИ ОБНАРУЖЕНИЯ ВРЕДОНОСНЫХ ПРОГРАММ

3.1 Исследование системных вызовов и разработка экспериментальный выборки

В настоящее время существует множество средств защиты информации в мобильных устройствах, работающих на программном и аппаратном уровнях. Современные системы защиты информации находятся на высоком уровне и демонстрируют хорошие результаты. Мобильные устройства применяются для широкого круга задач и являются удобным средством для решения многих вопросов. Этим обусловлена высокая скорость роста функциональных возможностей мобильных устройств. На фоне данной тенденции существует проблема комплексной защиты информации. Злоумышленники имеют сегодня в своем арсенале множество различных средств и активно используют их для достижения своих противоправных целей. В ОС для мобильных устройств эффективно могут использоваться хорошо зарекомендовавшие себя “классические” методы. Они основаны на сигнатурном анализе и эффективно работают при наличии соответствующей сигнатуры в соответствующей базе, но для того чтобы она там появилась, необходимо определенное время, что может послужить ключевым моментом для применения новых вредоносных программ. Вышеизложенное заставляет разработчиков в сфере ОС для мобильных устройств проводить анализ поведенческого характера для обнаружения вредоносных программ. Разработка комплексных средств защиты позволяет обнаруживать вредоносные программы как на уровне поведения, так на сигнатурном уровне.

Основой для защиты информации в ОС для мобильных устройств является система разрешений, которая реализована таким образом, чтобы предоставлять пользователю запрашиваемые им разрешения для работы его программы на этапе установки. Разрешения в системе имеют фиксированное значение и количество, могут быть классифицированы по уровню опасности. По числу и типу

разрешений можно сделать вывод о деятельности самой программы, которая выполняет запрос из списка разрешений. Во второй главе диссертационной работы была построена модель поведенческого характера вредоносной программы в ОС для мобильных устройств на основе скрытой марковской модели с целью выявления основных свойств, присущих поведению вредоносных программ, а также был выполнен анализ всех типов разрешений, представленных в приложении А. На основании экспериментальной информации получены векторы поведения как вредоносных, так и безопасных программ. Необходимо выполнить комплексный анализ методов классификации данных с целью определения наилучшего из них для моделирования системы обнаружения вредоносных программ. Под вредоносной программой понимается программа, предназначенная для осуществления несанкционированного доступа к информации и (или) воздействия на информацию или ресурсы информационной системы [34]. Разработанные векторы были сформированы по экспериментальным данным и использовались как для классических методов классификации, так и для нейросетевых в качестве обучающей и тестовой выборок, представленных в таблице 3.1.

Таблица 3.1 представляет собой набор признаков программ, заданных в бинарной форме (0 – отсутствует, 1 – присутствует), и включает в себя 100 векторов поведения программ. Столбцы с 1 по 10 содержат информацию о выявленных признаках в процессе анализа множества вредоносных программ. С 10 по 162 – список всех разрешений. Столбцы 163 и 164 содержат значения, полученные путем анализа используемых системных процессов в ОС для мобильных устройств. Для получения этих данных был выполнен анализ поведения вредоносных и безопасных программ с точки зрения системных процессов на эмуляторе с запущенной ОС Android.

Таблица 3.1 – Экспериментальная выборка

	1	2	3	4	5	6	7	8	9	10	…	162	163	164	165
1	0	0	0	0	0	0	0	0	0	0	…	1	40	70	ok
2	0	0	0	0	0	0	0	0	0	0	…	0	0	11	ok
3	1	0	0	0	0	0	0	0	0	0	…	0	95	10	virus
4	1	1	1	1	1	1	1	1	1	1	…	1	41	0	virus
5	1	1	0	0	0	0	0	0	1	1	…	1	94	9	virus
6	1	1	0	1	0	0	0	0	1	0	…	0	93	8	virus
7	1	1	1	1	1	1	1	1	1	0	…	0	92	7	virus
8	1	1	0	0	0	0	0	0	1	0	…	0	91	6	virus
9	1	1	0	0	0	0	0	0	1	0	…	0	90	5	virus
10	0	0	1	1	1	1	1	1	0	0	…	1	39	69	ok
11	0	0	1	1	1	1	1	1	0	0	…	1	38	68	ok
12	0	0	1	1	1	1	1	1	0	0	…	0	37	67	ok
13	1	1	0	0	0	0	0	0	1	0	…	0	36	66	ok
14	1	1	0	0	0	0	0	0	1	0	…	0	89	4	virus
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
95	1	1	0	0	0	0	1	1	0	0	…	0	42	12	virus
96	1	0	0	0	0	1	1	0	1	0	…	0	41	11	virus
97	0	0	0	0	0	1	1	1	0	0	…	0	41	10	ok
98	0	0	1	0	0	0	0	0	0	0	…	1	39	9	ok
99	0	0	0	0	0	0	0	0	0	0	…	1	38	8	ok
100	0	0	1	0	0	0	0	0	0	0	…	0	36	7	ok

Анализ проводился в четыре этапа:

1) поиск и инсталляция вредоносных и безопасных программ в ОС для мобильных устройств с целью выполнения анализа поведения с точки зрения системных вызовов;

2) отслеживание процесса и его идентификатора – PID с помощью монитора процессов, а также внутренних команд в эмуляторе терминала ОС Linux (команда ps, top) [23];

3) каждая программа запускается в строгом соответствии с временными рамками, а также имитируются действия для проверки ее основных функций;

4) в терминале командой strace –c –p PID, где PID – номер присвоенного идентификатора, была получена общая статистика обращений к системным вызовам для каждой программы.

Были рассмотрены 30 вредоносных и 15 безопасных программ, результаты проделанной работы представлены на рисунках 3.1–3.2.

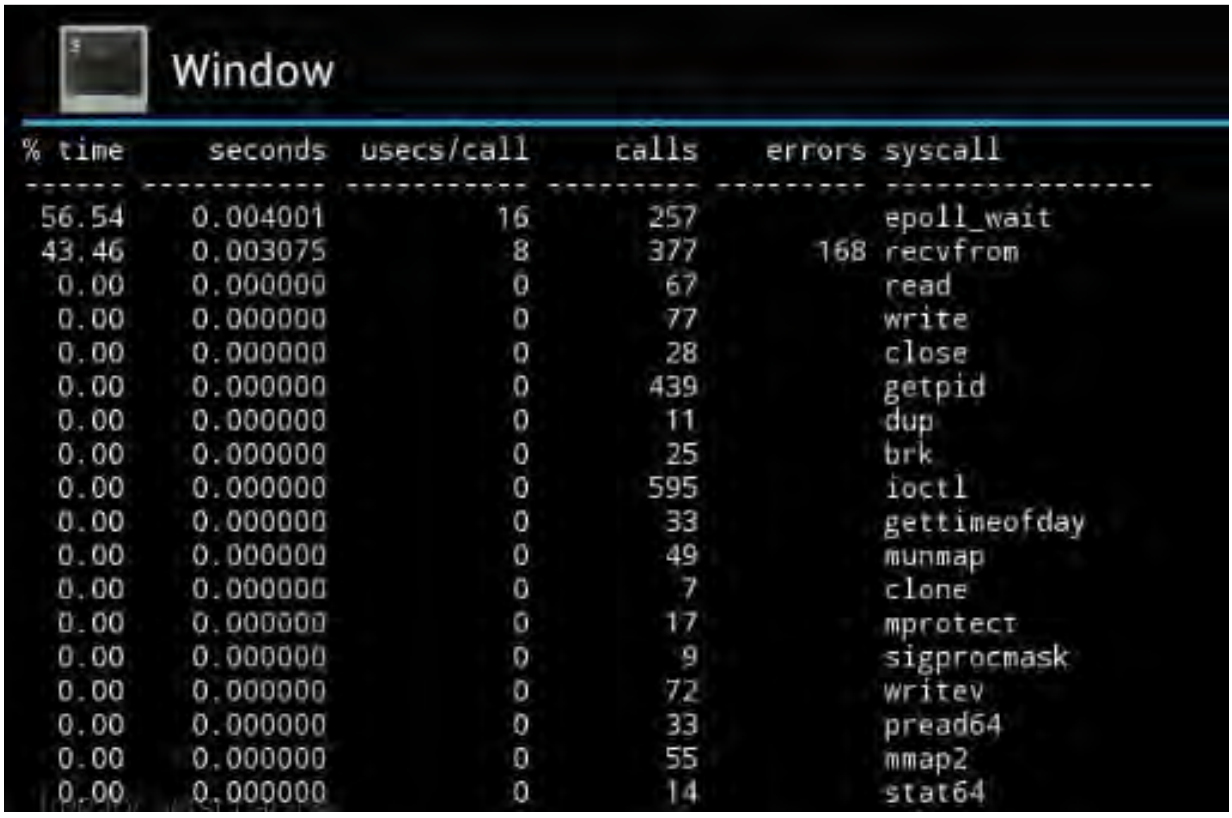


Рисунок 3.1 – Общая таблица системных вызовов безопасной программы

Дальнейший анализ позволил установить, что вредоносные программы используют системный вызов sendto, который отправляет сообщения в сокет (отправляет большое количество информации без предварительной установки соединения) до 95 % общего времени, и системный вызов epoll_wait, который

ожидает события ввода/вывода об изменившихся файловых дескрипторах с произошедшими событиями epoll_wait до 10 % всего времени. Безопасная программа использует sendto до 40 % и epoll_wait до 70 % всего времени, что является явным признаком для возможности проведения динамического анализа (рисунок 3.3–3.4). Полный список системных вызовов и пояснения их назначения находятся в документации для ОС Linux [23].

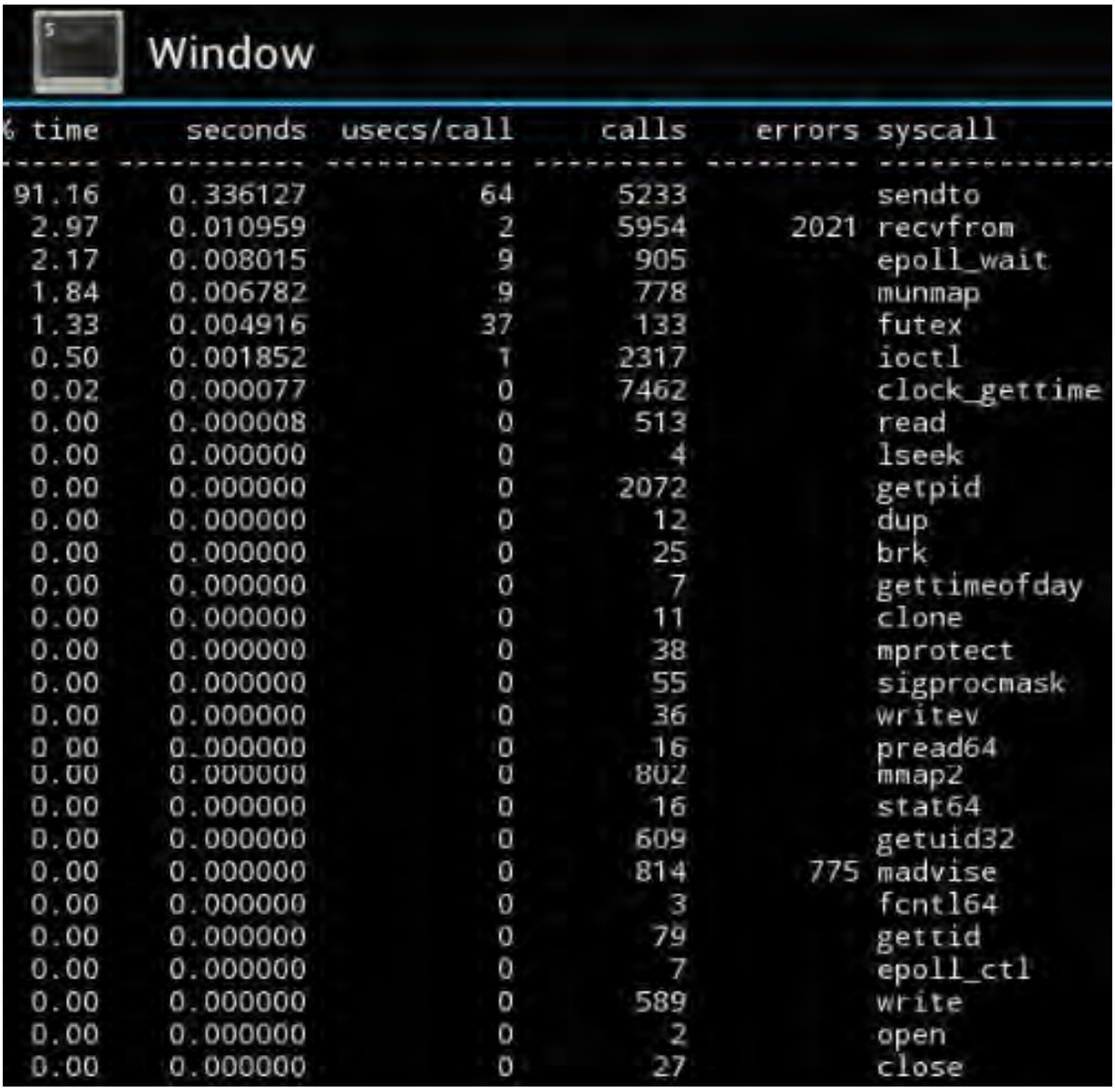


Рисунок 3.2 – Общая таблица системных вызовов вредоносной программы

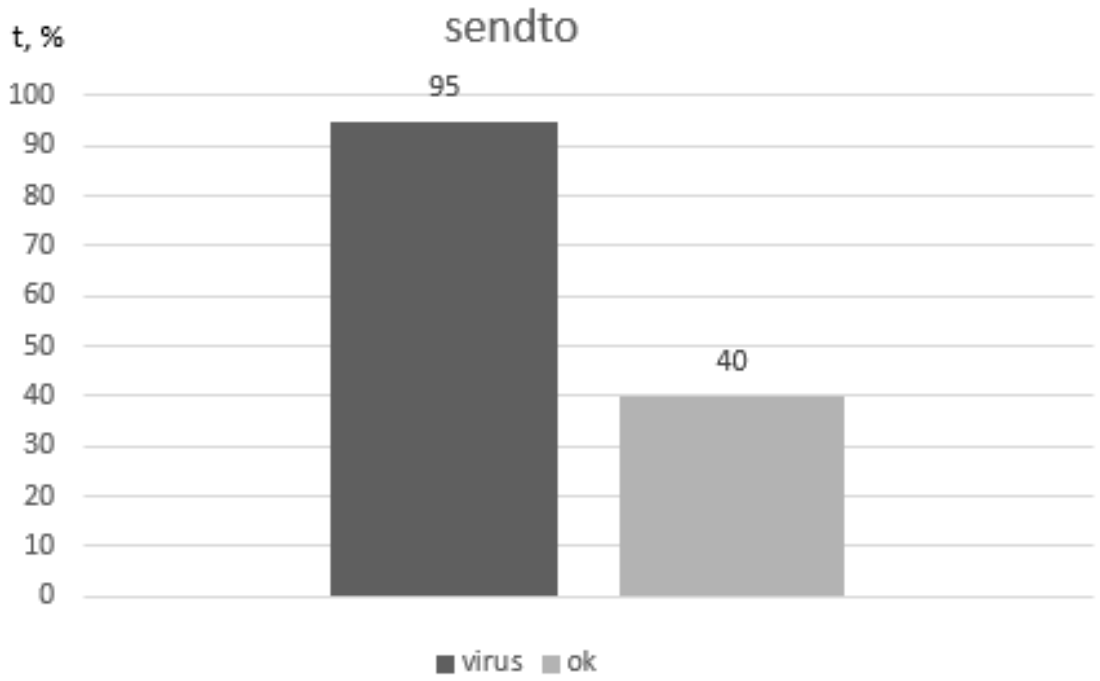


Рисунок 3.3 – Время обращения к системному вызову sendto

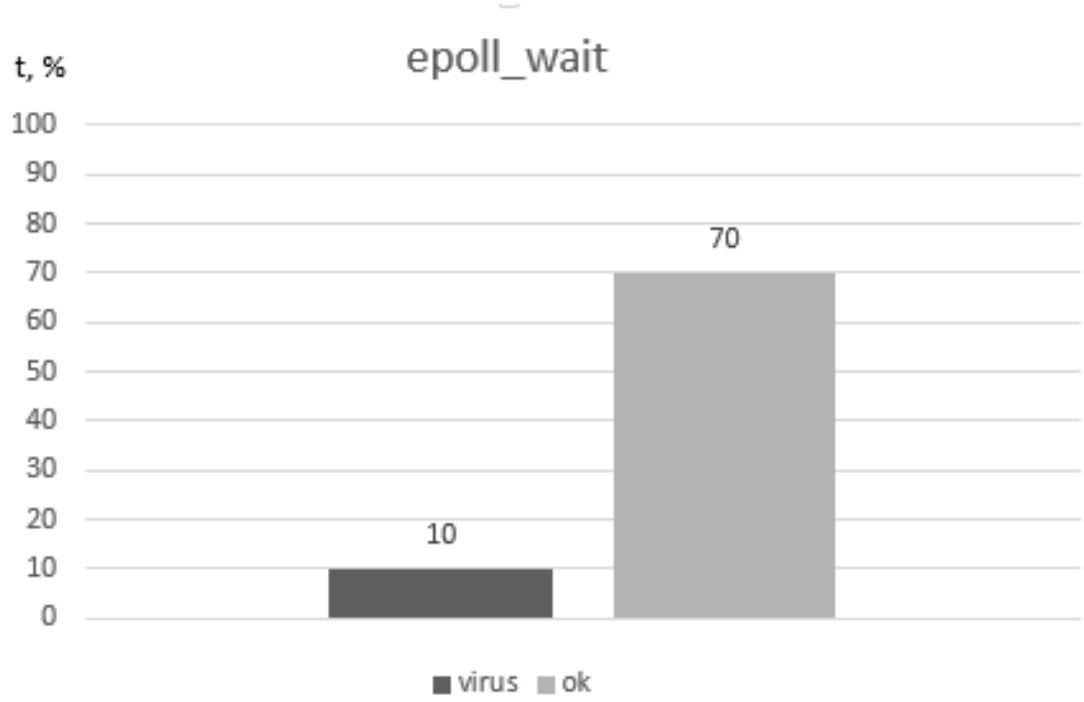


Рисунок 3.4 – Время обращения к системному вызову epoll_wait

Признаком, описывающим поведение того или иного типа программ, является резкое увеличение количества времени обращения к системным

вызовам, как отображено на рисунках. Промежутки между временем обращения можно использовать в качестве признаков, характеризующих поведение вредоносной и безопасной программы. Для вредоносной программы, работающей с системным вызовом sendto, был взят диапазон от 45 до 95, с epoll_wait от 0 до 10. Для безопасной программы соответственно от 0 до 40 и от 15 до 70. Таким образом, были сформированы значения 164 и 165 в экспериментальной выборке (таблица 3.1).

Векторы состоят из признаков, присущих поведению программ, заданных на основе разрешений, системных вызовов и свойств, присущих поведению того или иного типа программ. По мере приближения к нижнему вектору вносились помехи с целью оценки качества обнаружения при разных уровнях помех.

Для выбора оптимального метода классификации были применены классические и нейросетевые методы, а также машина опорных векторов с целью выявления количественных и качественных показателей для реализации модели уникального классификатора вредоносных программ.

3.2 Постановка задачи классификации разработанной экспериментальной выборки

Пусть $\boldsymbol{X}$ – множество объектов (программ), $\boldsymbol{Y}$ – множество номеров (virus, ok) кластеров. Классификация осуществляется на основе двух классов: virus и ok. В качестве метрики выбрано евклидово расстояние между объектами:

$$\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{x}') = (\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} (\boldsymbol{x} - \boldsymbol{x}')^2)^{1/2}.$$

Задана конечная экспериментальная выборка объектов:

$$\boldsymbol{X}^{\boldsymbol{m}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{\boldsymbol{m}}\} \subset \boldsymbol{X}.$$

Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких

по метрике $\boldsymbol{p}$, а объекты разных кластеров существенно различались. При этом каждому объекту $\boldsymbol{x_i} \in \boldsymbol{X^m}$ приписывается номер кластера $\boldsymbol{y_i}$ [32, 37].

Решение задачи: требуется определить функцию $\boldsymbol{a}: \boldsymbol{X} \to \boldsymbol{Y}$, которая любому объекту $\boldsymbol{x} \in \boldsymbol{X}$ ставит в соответствие номер кластера $\boldsymbol{y} \in \boldsymbol{Y}$. Множество $\boldsymbol{Y}$ в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.

В качестве критерия точности и качества работы классификаторов будем использовать следующую формулу:

$$ОК = \frac{ЧО \times 100\%}{ЧН},$$

где ОК – общий процент как вредоносных, так и безопасных программ ошибки классификации;

ЧО – число ошибок классификации;

ЧН – суммарное число наблюдений.

3.3 Классификация экспериментальной выборки классическими методами

Используем классические методы для классификации экспериментальной таблицы признаков векторов программ. Целью данной классификации является разбиение векторов программ на классы. Каждый класс соответствует типу программ: вредоносные программы и безопасные.

В результате классификации методом иерархической кластеризации была построена дендрограмма, отображающая результаты проделанной работы.

Дендрограмма начинается слева для каждого вида программы в своем отдельном кластере. По мере перемещения вправо экспериментальные данные объединяются и формируют кластеры согласно формуле

$$K_\eta([i,j],k) = [\frac{(n_i K(i,k)^\eta) + (n_j K(j,k)^\eta)}{n_i + n_j}]^{\frac{1}{\eta}}, -\infty \le \eta \le +\infty,$$

где $[i, j]$ – группа из двух кластеров;

k – кластер, в котором выполняется поиск сходства с указанной группой;

n_i – количество элементов в кластере i;

n_j – количество элементов в кластере j.

Результат работы метода иерархической кластеризации приведен на рисунке 3.5.

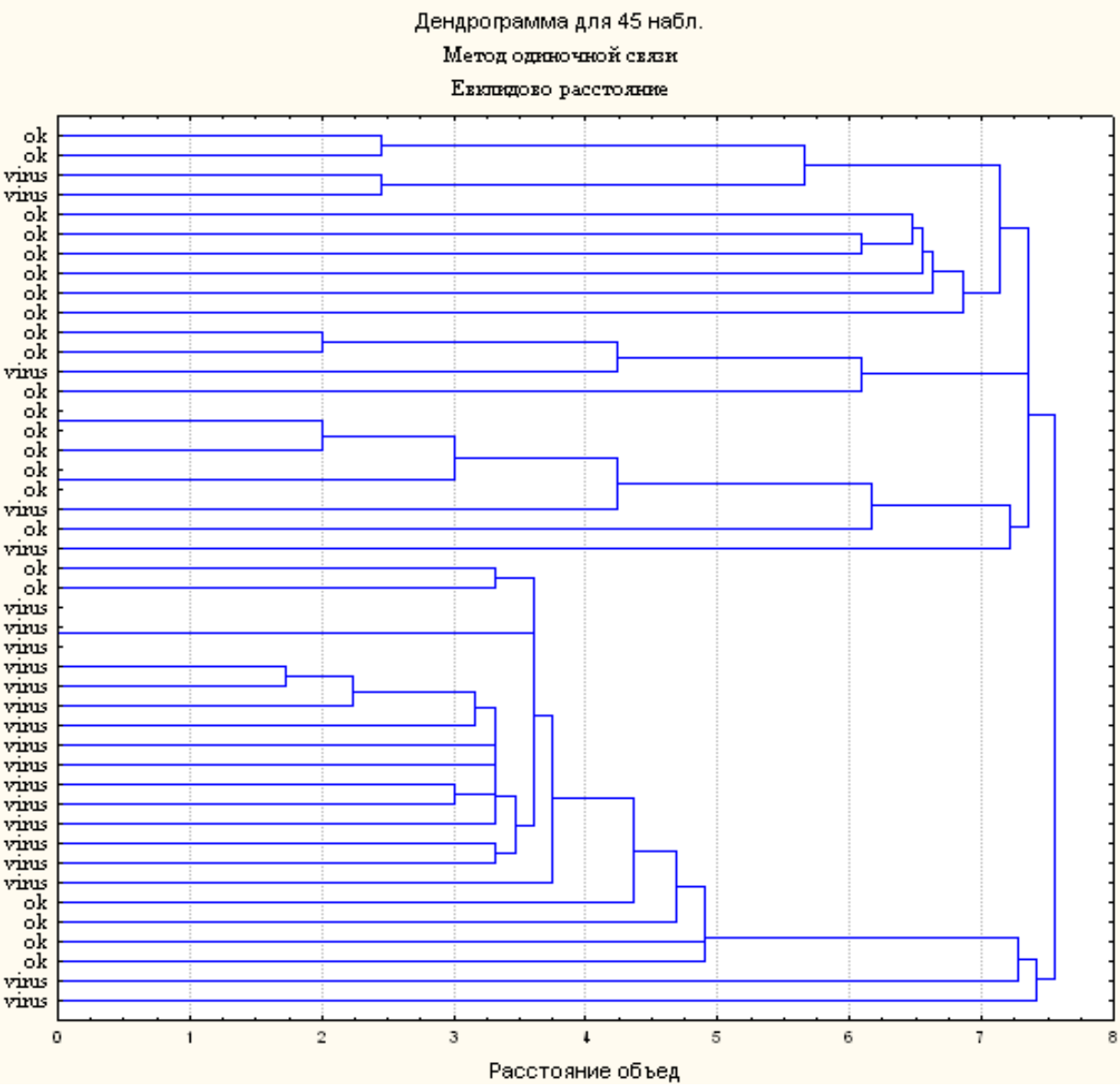


Рисунок 3.5 – Результат работы метода иерархической кластеризации

Анализ дендрограммы позволяет констатировать следующее: экспериментальные данные по итогам решения задачи были поделены на два

множества. Каждое множество содержит как безопасные, так и вредоносные программы. Количество ошибок первого рода, когда множество безопасных программ содержит вредоносные, равно 22,7 %, количество ошибок второго рода, когда множество вредоносных программ функционирует исправно (вредоносные программы не обнаружены), равно 26,08 %.

Процесс классификации можно отследить по диаграмме расстояния объединения, по шагам представленной на рисунке 3.6.

Диаграмма отображает процесс классификации пошагово. Каждый шаг соответствует вектору программ: 45 наблюдений равно 45 шагам. По диаграмме видно, что по мере увеличения числа шагов уменьшается расстояние между кластерами, это обусловлено увеличением количества помех по мере приближения к 45-му шагу.

Кластеризация методом k-средних выполнена по двум кластерам, так как мы имеем два множества. Данный метод стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2, \; i = 1, 2, 3, \ldots, k,$$

где k – число кластеров равное двум;

S_i – полученные кластеры;

μ_i – центры масс векторов $x_j \in S_i$.

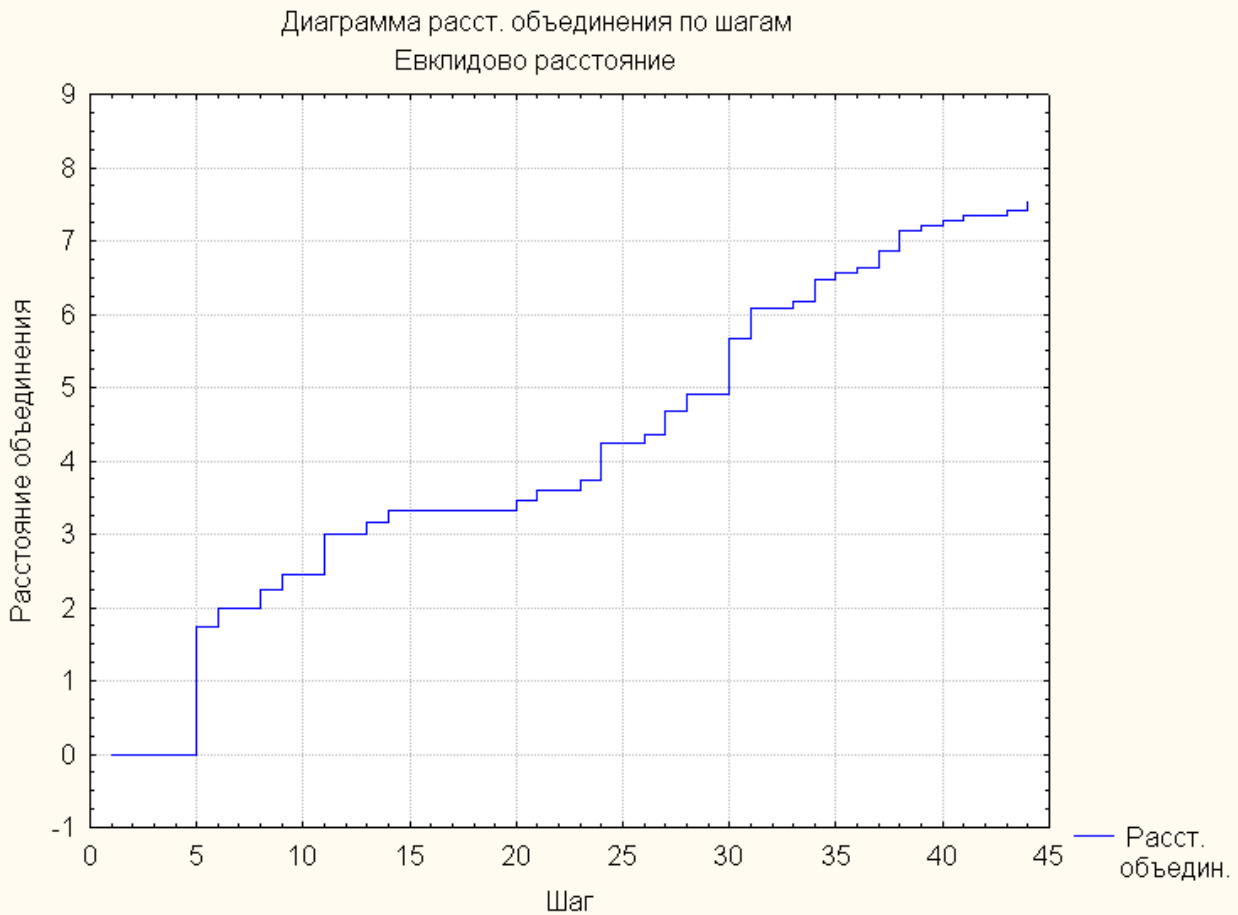


Рисунок 3.6 – Диаграмма расстояния объединения по шагам

В результате проделанной классификации сформировалось два кластера: множество вредоносных программ и множество безопасных программ. Результаты работы приведены в таблице 3.2 для метода k-средних.

Таблица 3.2 – Результаты классификации методом k-средних

Кластер 1	Кластер 2
1	2
ok	ok
virus	virus
virus	virus
virus	ok

Окончание таблицы 3.2

1	2
virus	ok
virus	ok
virus	ok
virus	virus
virus	virus
virus	ok
virus	ok
virus	ok
virus	ok
virus	virus
virus	ok
ok	ok
virus	ok
virus	ok
virus	ok
ok	ok
ok	ok
ok	ok
ok	–

В процессе классификации было установлено, что кластеры содержат вредоносные и безопасные программы.

Качество распознавания классическими методами (иерархической кластеризации и k-средних) одинаковое: ошибки I рода – 26,08 %, ошибки II рода – 22,7 %. Применение факторного и дискриминантного анализа показало в среднем такой же уровень ошибок в задаче классификации, что говорит о невысокой точности работы методов. Дальнейший анализ этих методов

показывает, что с увеличением количества помех (внешних и внутренних факторов в виде аддитивной или мультипликативной составляющей экспериментальной выборки) точность классификатора значительно падает.

3.4 Классификация экспериментальной выборки нейросетевыми методами

При классификации нейронными сетями образец представляется в виде вектора $\boldsymbol{x}$, принадлежащего к N-мерному пространству $\boldsymbol{R}^{N}$, компоненты которого представляют собой различные характеристики образца. Классификатор относит объект $\boldsymbol{x}^{k}$ к тому или иному классу C в соответствии с определенным разбиением N-мерного пространства входов.

Нейронные сети, в отличие от статистических методов многомерного классификационного анализа, базируются на параллельной обработке информации и обладают способностью к самообучению, то есть получают обоснованный результат на основании данных, которые не встречались в процессе обучения. Эти свойства позволяют нейронным сетям решать сложные задачи, которые в настоящее время считаются трудноразрешимыми [58, 59].

Для эффективного использования нейронных сетей необходимо наличие достаточного объема обучающей выборки, используя которую, нейронную сеть можно обучить. Экспериментальные данные, приведенные в таблице 3.1, были использованы в качестве обучающей и тестовой выборки для нейронных сетей. Для обучения использовались первые 68 векторов. Для тестирования – остальные 32.

Выбор архитектуры нейронной сети выполняется в соответствии с типом решаемой задачи. Для классификации подходят: линейная нейронная сеть, многослойный персептрон, радиально-базисная функция, вероятностные нейронные сети и сети Кохонена [71, 80]. Для выполнения классификации были выбраны радиально-базисная функция, многослойный персептрон и линейная нейронная сеть.

Для обучения выбранных нейронных сетей применялся метод обратного распространения ошибки. Данный алгоритм обеспечивает настройки весов с учетом многослойной структуры сети. Ошибка обучения на выходе нейронной сети распространяется в обратном направлении к скрытым слоям. Для нейронов выходного слоя величина ошибки вычисляется просто как разность между ожидаемым и реальным выходным значением.

В результате обучения нейронных сетей были построены графики обучения, приведенные на рисунках 3.7–3.9.

Число эпох составило 110, скорость обучения 0.1. На графике отображена зависимость ошибки обучения от числа эпох для многослойного персептрона. Можно отметить, что при приближении к 50-й эпохе ошибка значительно уменьшилась, и достигнута минимальная ошибка при приближении к 100-й эпохе.

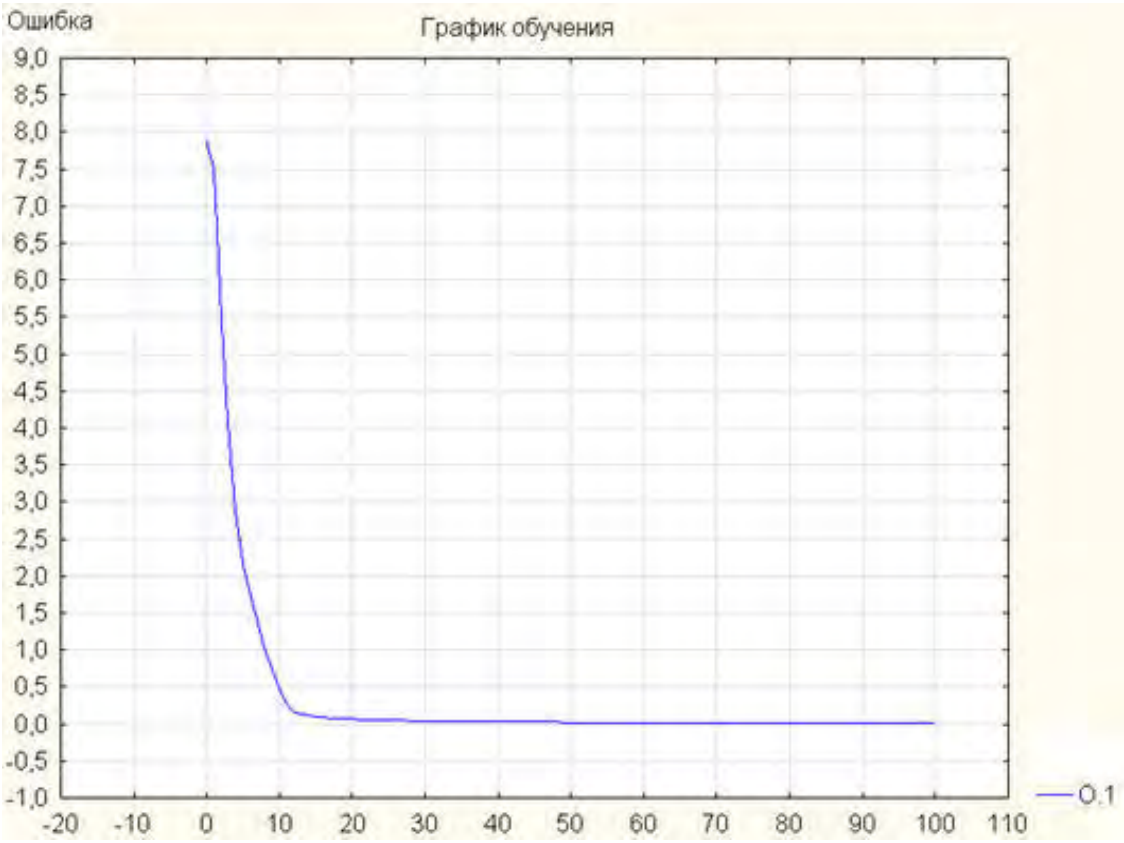


Рисунок 3.7 – График обучения многослойного персептрона

Рисунок 3.8 – График обучения радиально-базисной функции

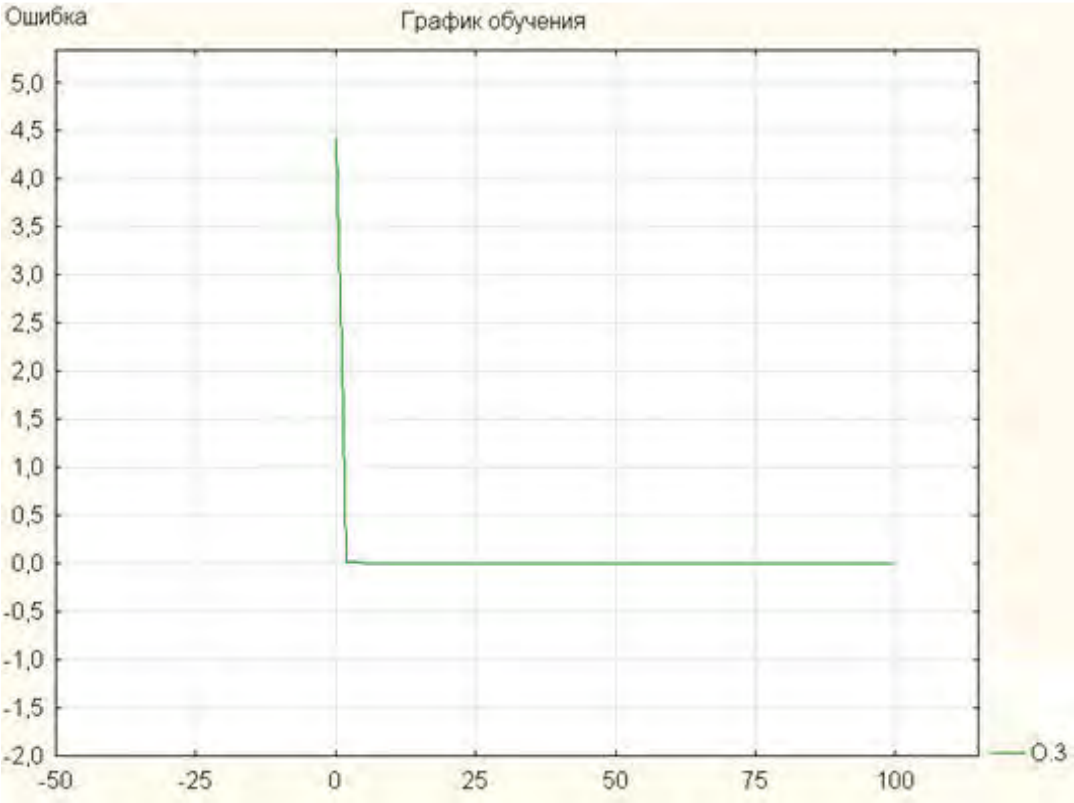


Рисунок 3.9 – График обучения линейной нейронной сети

Выбор функции активации осуществляется в зависимости от задачи, удобства программно-аппаратной реализации, а также алгоритма обучения. Аргумент функции активации каждого скрытого узла сети радиальной базисной функции представляет собой евклидову норму между входным вектором и центром радиальной функции. Аргумент функции активации каждого скрытого узла сети многослойного персептрона является скалярным произведением входного вектора и вектора синаптических весов данного нейрона. Аргумент функции активации для линейной нейронной сети представляет собой линейную дискриминантную функцию.

На рисунках 3.10–3.12 изображена выбранная архитектура построенных сетей с указанием активации каждого нейрона для того наблюдения, которое было задано. Интенсивность окраса нейронов соответствует их активациям, показывая визуальную индикацию активности каждой сети.

1. Многослойный персептрон – указаны активации каждого нейрона для наблюдения №1.

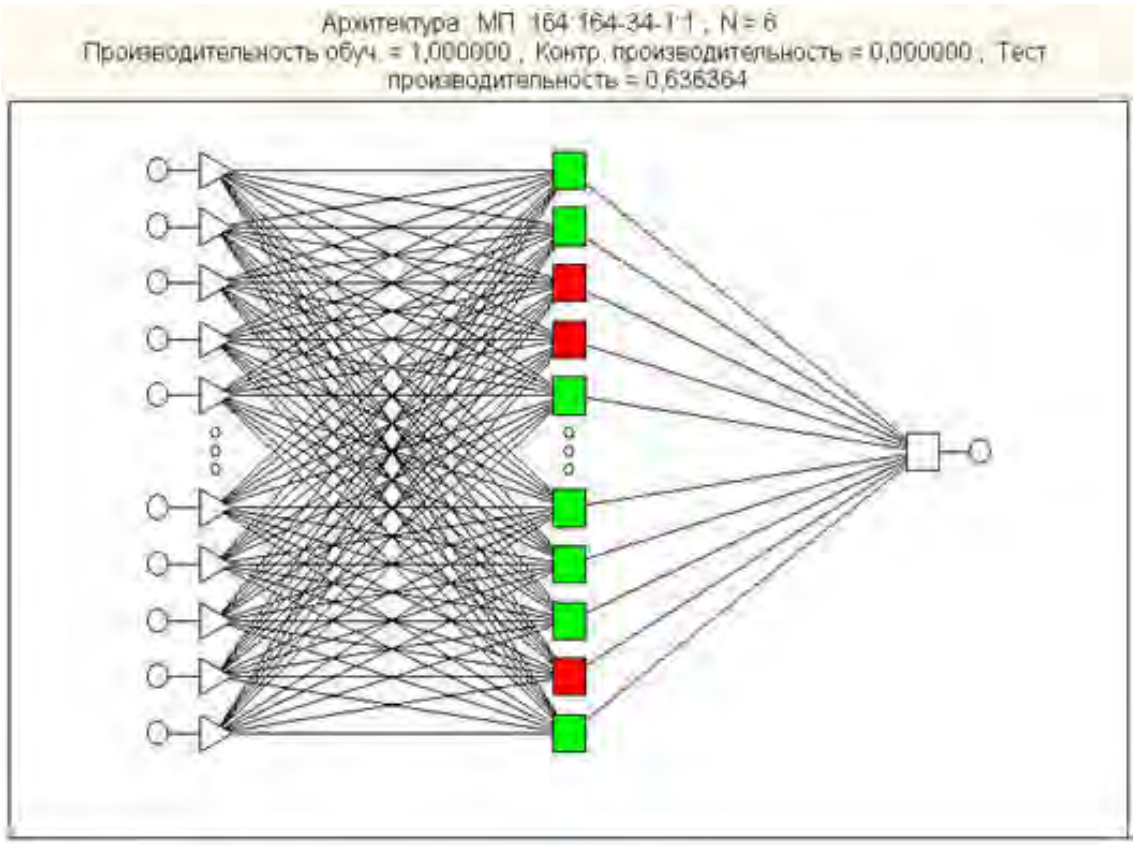


Рисунок 3.10 – Архитектура многослойного персептрона

2. Радиально-базисная функция – указаны активации каждого нейрона для наблюдения №1.

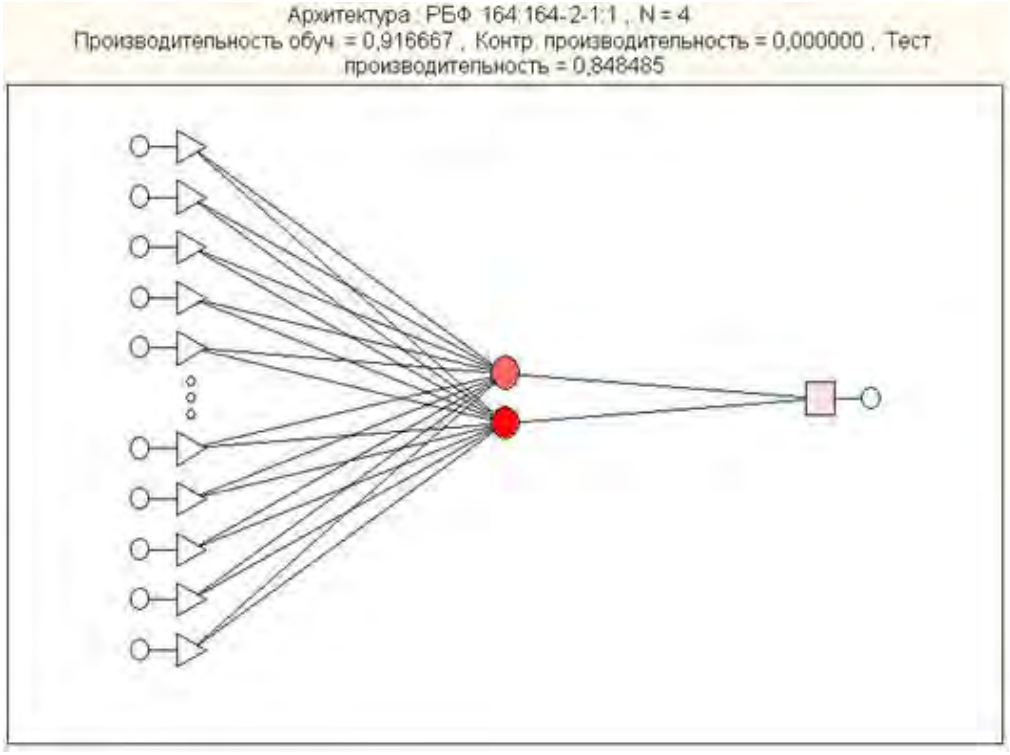


Рисунок 3.11 – Архитектура радиально-базисной функции

3. Линейная – указаны активации каждого нейрона для наблюдения №1.

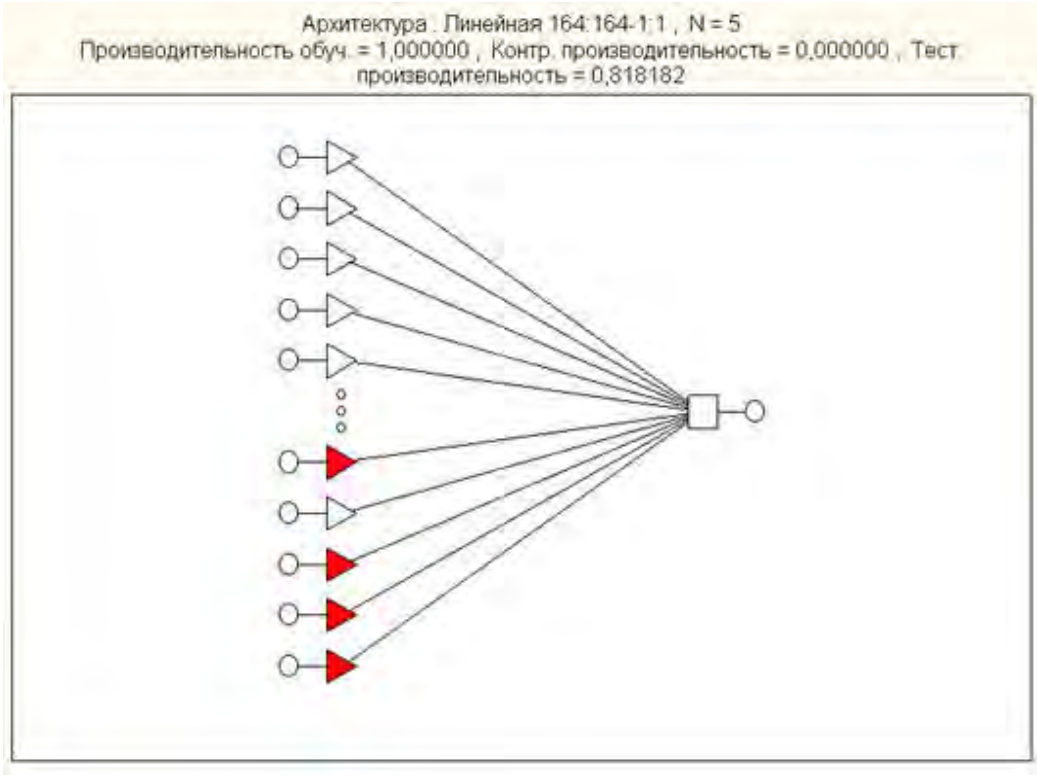


Рисунок 3.12 – Архитектура линейной сети

На рисунках также представлены: производительность обучения в целом, производительность обучения на контрольной выборке, производительность обучения на тестовой выборке, а также представлена информация о структуре нейронной сети (количество входов и скрытых слоев).

Результаты работы различных архитектур нейронных сетей приведены в таблице 3.3.

Таблица 3.3 – Сравнительный анализ результатов классификации нейронных сетей

№	Эталонная модель	РБФ (164-2-1)	Линейная (164-1)	МП (164-34-1)
1	2	3	4	5
68	ok	ok	virus	virus
69	virus	ok	virus	virus
70	virus	virus	virus	virus
71	virus	ok	virus	virus
72	ok	ok	ok	ok
73	ok	ok	ok	ok
74	ok	ok	ok	ok
75	ok	ok	ok	ok
76	virus	virus	virus	virus
77	virus	virus	virus	virus
78	virus	virus	virus	virus
79	virus	ok	virus	virus
80	ok	ok	ok	ok
81	ok	ok	ok	virus
82	virus	virus	virus	virus
83	virus	virus	virus	virus

Окончание таблицы 3.3

1	2	3	4	5
84	ok	ok	ok	virus
85	virus	virus	ok	virus
86	ok	ok	ok	ok
87	ok	ok	ok	virus
88	ok	ok	ok	virus
89	ok	ok	ok	virus
90	virus	virus	ok	virus
91	virus	virus	virus	virus
92	ok	ok	ok	virus
93	ok	ok	ok	virus
94	virus	virus	virus	virus
95	virus	virus	virus	virus
96	virus	virus	ok	virus
97	ok	ok	ok	virus
98	ok	virus	ok	virus
99	ok	ok	ok	virus
100	ok	virus	ok	virus

По результатам проделанной классификации была составлена таблица 3.4, в которой приведены результаты работы нейронных сетей.

Таблица 3.4 – Результаты классификации нейронными сетями

Результат	РБФ (164-2-1)		Линейная (164-1)		МП (164-34-1)	
	ok	virus	ok	virus	ok	virus
Всего	18	15	18	15	18	15
Правильно	16	12	16	11	6	15
Ошибочно	2	3	2	4	12	0
% правильных	88,9	80	88,8	73,3	33,3	100
% ошибочных	11	20	11,1	26,6	66,6	0

Также были рассмотрены нейронные сети, работающие с учетом режимов офлайн и онлайн:

- нейронная сеть Ворда, особенность архитектуры которой заключается в том, что ее скрытые слои нейронов разделены на блоки;
- модульная нейронная сеть представляет собой сеть, состоящую из аналогичных и, в определенной степени, независимых нейронных модулей;
- нейронная сеть прямого распространения представляет собой систему взаимодействующих адаптивных элементов – нейронов, каждый из которых выполняет определенное функциональное преобразование над сигналами;
- нейронная сеть прямого распространения с учетом временного окна равного 12 значениям;
- нейронная сеть Элмана представляет собой многослойный персептрон с введенными обратными связями, которые идут от выходов внутренних нейронов;
- рекуррентная нейронная сеть – сеть с обратными связями;
- персептрон;
- сеть Кохонена – класс нейронной сети, в качестве основного элемента которой выступает слой Кохонена. Он состоит из адаптивных линейных сумматоров. Обработка входных сигналов выполняется по правилу “победитель

получает всё”, то есть наибольший сигнал превращается в единичный, остальные обращаются в ноль.

В качестве функции активации для нейронных сетей была выбрана сигмоидная. Результаты классификации представлены в таблице 3.5.

Таблица 3.5 – Результаты классификации

Режим	Офлайн				Онлайн									
Тип нейронной сети	Ворда		Модульная		Прямого распространения		Прямого распространения (временное окно)		Элмана		Рекуррентная		Персептрон	
Классы	ok	virus	ok	virus	ok	virus	ok	virus	ok	virus	ok	virus	ok	virus
Всего	48	52	18	15	18	15	18	15	18	15	18	15	18	15
Правильно	0	52	7	15	12	14	6	12	9	14	14	5	0	10
Ошибочно	48	0	11	0	6	1	12	3	9	1	4	10	0	5
% правильных	0	100	38,9	100	66,7	92,9	33,4	80	50	92,9	77,8	33,4	100	73,4
% ошибочных	100	0	61,1	0	33,3	7,1	66,6	20	50	7,1	22,2	66,6	0	66,7

Обучение проводилось на первых 67 векторах экспериментальной выборки. Остальные 33 вектора выступали в качестве тестовой выборки (таблица 3.6). Сеть Кохонена выполнила разделение векторов на 5 классов: первый, третий и пятый относятся к классу вредоносных программ – virus. Второй и четвертый – к классу безопасных программ – ok (таблица 3.7).

Структура нейронной сети Кохонена представлена на рисунке 3.13.

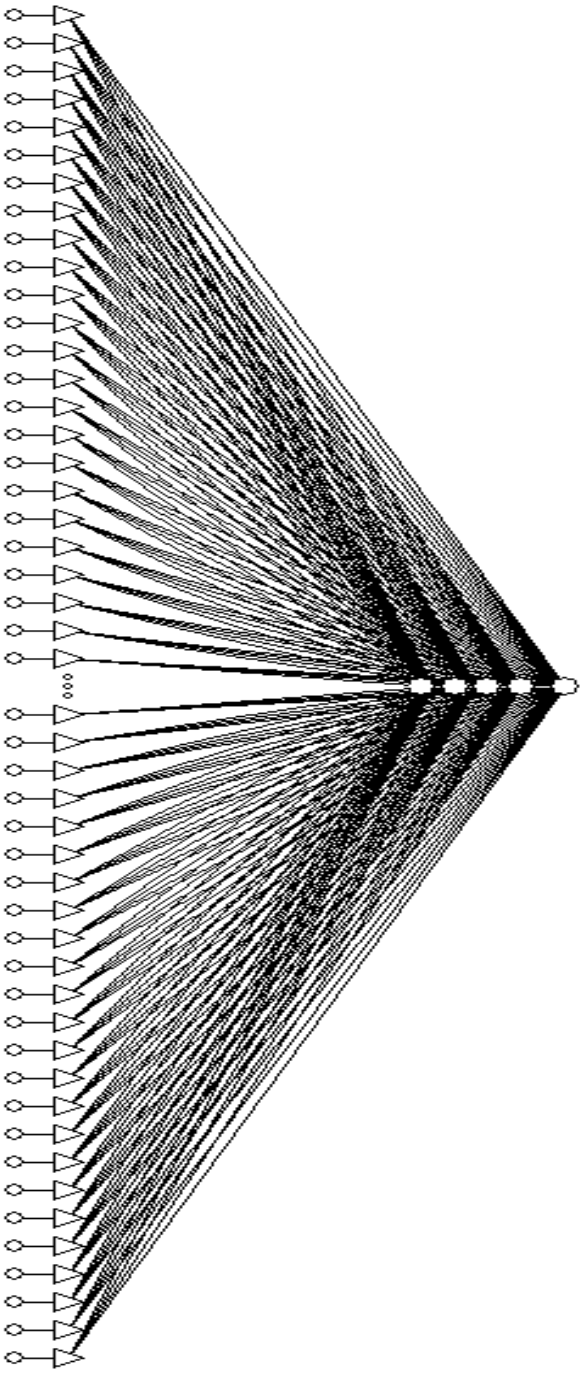


Рисунок 3.13 – Структура сети Кохонена

Таблица 3.6 – Результат классификации сетью Кохонена

№	Тип программы	Класс	№	Тип программы	Класс
68	ok	4	85	virus	5
69	virus	5	86	ok	3
70	virus	4	87	ok	3
71	virus	2	88	ok	3
72	ok	2	89	ok	3
73	ok	4	90	virus	5
74	ok	2	91	virus	1
75	ok	3	92	ok	5
76	virus	3	93	ok	3
77	virus	2	94	virus	5
78	virus	2	95	virus	1
79	virus	5	96	virus	1
80	ok	4	97	ok	5
81	ok	4	98	ok	5
82	virus	5	99	ok	1
83	virus	5	100	ok	1
84	ok	4	-	-	-

Таблица 3.7 – Результаты сети Кохонена

Результат	класс virus	класс ok
Всего	15	18
Правильно	11	7
Ошибочно	4	11
% правильных	73,3	38,8
% ошибочных	26,6	61,1

Таким образом, сеть Кохонена отнесла 4 вредоносных программы в классы два и четыре, принадлежащие к безопасным программам, 11 безопасных программ – к классам вредоносных программ (первый, третий и пятый).

Ошибки классификации нейронными сетями присутствуют по всей тестовой выборке, но увеличиваются с ростом количества помех в векторе.

Анализ решаемой задачи классификации на основе нейронных сетей показывает, что лучший результат показала нейронная сеть на основе радиально-базисной функции (таблица 3.4). В таблицах 3.4 и 3.5 отображены результаты проделанных экспериментов, ошибки первого и второго рода.

Таким образом, классификация нейронными сетями показала лучший результат по сравнению с классическими методами классификации в условиях с увеличенным количеством помех, но при сильном увеличении помех более чем на 70 % нейронные сети совершают ошибки классификации.

3.5 Классификация экспериментальной выборки машиной опорных векторов

Машина опорных векторов классифицирует методом построения нелинейной плоскости, выполняющей разделения классов. Благодаря особенностям природы пространства признаков, в котором строятся границы решения, машина опорных векторов обладает высокой степенью гибкости при решении задачи классификации различного уровня сложности [84].

Машина опорных векторов широко используется в сфере защиты информации и является очень популярным методом. Во множестве работ российских авторов, например, И. В. Машкиной, В. И. Васильева, С. С. Валеева [22] разрабатывались модели обнаружения сигнатур атак на основе метода опорных векторов [16, 18, 19], также разрабатывались модели обнаружения аномалий в сети на основе метода опорных векторов [54] и т.д. В. Д. Котов в своей работе применял машину опорных векторов в качестве средства для повышения эффективности обнаружения вредоносных интернет-страниц [62],

Л. А. Свечников занимался разработкой интеллектуальной системы обнаружения атак [73], . Машина опорных векторов широко используется для построения систем защиты информации в среде ОС Windows, но в новой мобильной ОС типа Android на базе ядра Linux в российском сегменте работы не проводились, что дает повод применить эффективную и популярную машину опорных векторов для увеличения эффективности обнаружения вредоносных программ в ОС типа Android. Много зарубежных работ посвящено разработке интеллектуальных систем обеспечения защиты информации в мобильных ОС типа Android. Анализ работ [98, 102, 108, 111] показал, что область исследования является актуальной и машина опорных векторов применяется в данном направлении. В основе машины опорных векторов лежит нейронная сеть, что позволяет проводить обучение. В своем эксперименте я применил машину опорных векторов в задаче классификации. Ставится задача классификации программ на два класса: virus и ok. Машину опорных векторов можно построить на различных типах ядер. В данном случае было задействовано ядро радиально-базисной функции, так как нейронная сеть на основе радиально-базисной функции в задаче классификации показала лучшие результаты:

$$\varphi = \exp(-\gamma | \boldsymbol{x_i} - \boldsymbol{x_j} |^2).$$

Машина опорных векторов обучена на первых 68 и тестировалась на остальных 32 значениях векторов программ экспериментальной выборки, приведенных в таблице 3.1. Обучение машины опорных векторов включает в себя минимизацию функции ошибки:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i,$$

при ограничениях:

$$y_i(w^T \varphi(x_i) + b) \geq 1 - \xi_i,$$

$$\xi_i \geq 0, i = 1, \dots, N.$$

Ядро используется для преобразования данных из входных (независимых) в пространство признаков.

Результаты, приведенные в таблице 3.8, описывают результат классификации обученной машиной опорных векторов.

Таблица 3.8 – Результаты классификации

Номер вектора	Эталонная модель	Результат классификаций
1	2	3
69	ok	ok
70	virus	virus
71	virus	virus
72	virus	virus
73	ok	ok
74	ok	ok
75	ok	ok
76	ok	ok
77	virus	virus
78	virus	virus
79	virus	virus
80	virus	virus
81	ok	ok
82	ok	ok
83	virus	virus
84	virus	virus
85	ok	ok
86	virus	virus
87	ok	ok

Окончание таблицы 3.8

1	2	3
88	ok	ok
89	ok	ok
90	virus	virus
91	virus	virus
92	ok	virus
93	ok	ok
94	virus	virus
95	virus	virus
96	virus	virus
97	ok	virus
98	ok	virus
99	ok	virus
100	ok	virus

В результате проделанной работы точность обучения составила 100 %. Количество составленных опорных векторов – 8 по 4 в каждом классе, из них один на границе.

По гистограмме (рисунок 3.14) можно заметить, что все векторы разделились на два класса – virus и ok, включая обучающую и тестовую выборку. В класс virus попало 22 наблюдения, в класс ok попало 23 наблюдения.

Машина опорных векторов показала лучшие результаты по сравнению с нейронными сетями и классическими методами. Результаты приведены в таблице 3.9.

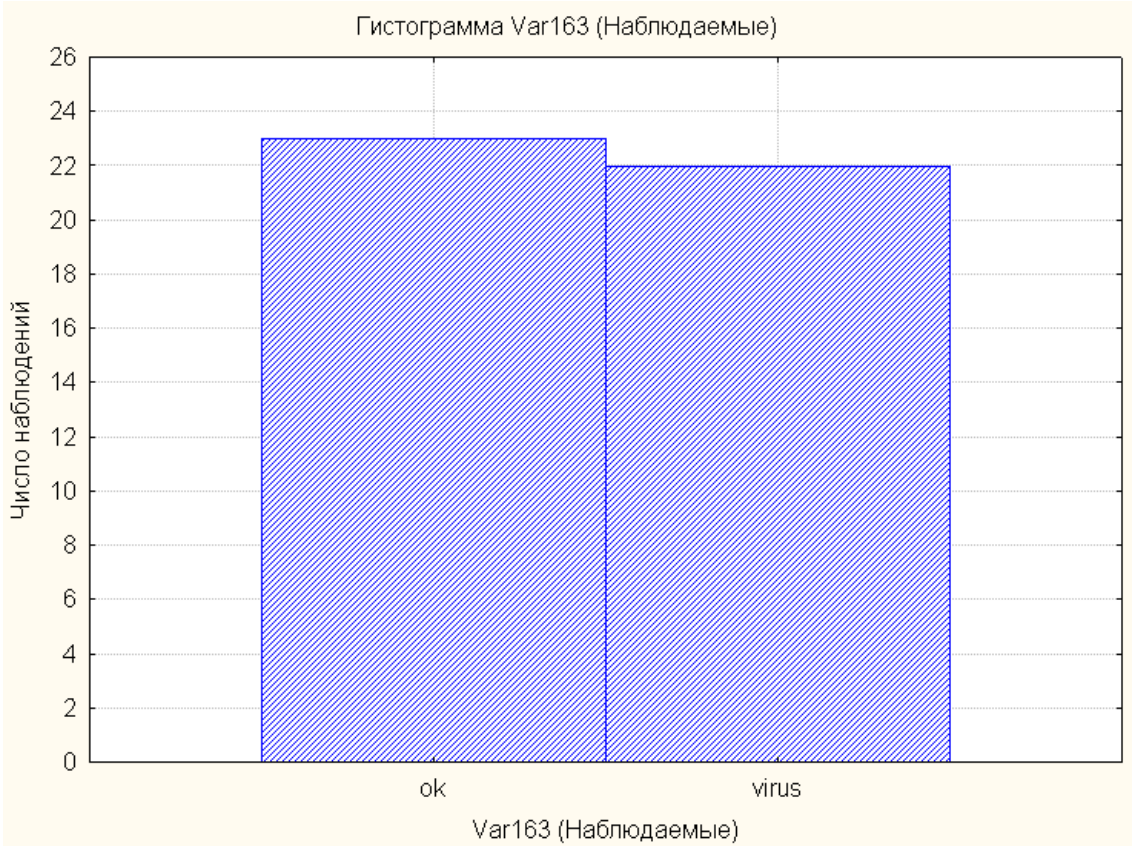


Рисунок 3.14 – Гистограмма классификации машиной опорных векторов

Таблица 3.9 – Сравнительная таблица результатов классификации

Результат	Иерархи-ческая классифи-кация		K-средних		РБФ (164-2-1)		Линейная (164-1)		МП (164-34-1)		Машина опорных векторов	
Класс	ok	virus	ok	virus	ok	virus	ok	virus	ok	virus	ok	virus
Всего	22	23	23	22	18	15	18	15	18	15	18	15
Правильно	17	17	17	17	16	12	16	11	6	15	13	15
Ошибочно	5	6	6	5	2	3	2	4	12	0	5	0
% правильных	77,2	73,9	73,9	77,2	88,9	80	88,8	73,3	33,3	100	72,3	100
% ошибочных	22,8	26,1	26,1	22,8	11	20	11,1	26,6	66,6	0	27,7	0

Всего в классификации участвовали 32 вектора тестовой выборки. Из них 18 – ok и 14 – virus. Классификация в класс ok была проведена с ошибками. Ошибочно выполнена классификация 5 наблюдений, процент правильно классифицированных составил 72,3. Количество ошибок I рода – 0 %, ошибок II рода – 27,7 %. В классе virus классификация была выполнена безошибочно и таким образом, что процент правильных составил 100 %.

Сравнивая результаты экспериментов, полученные классическими методами, нейронными сетями и машиной опорных векторов, можно сделать следующий вывод: машина опорных векторов обладает встроенной способностью решать задачу классификации множеств, получаемый результат близок к оптимальному значению. Для того чтобы нейронные сети (многослойный персептрон), обучаемые по алгоритму обратного распространения, достигли производительности машины опорных векторов, необходимо встроить в архитектуру знания о конкретной проблемной области и настроить множество параметров, что может быть трудновыполнимо в сложных условиях обучения.

На основе проделанной работы было предложено разработать систему обнаружения с применением машины опорных векторов для решения задачи классификации вредоносных программ. Для увеличения надежности и точности обнаружения была задействована нечеткая логика.

3.6 Система поддержки принятия решения на основе нечеткой логики для выполнения дополнительной классификации полученного результата машины опорных векторов

Объединение нечеткого логического вывода и экспертных оценок является одним из перспективных подходов к организации систем динамического анализа вредоносных программ, способствующих повышению надежности защиты информации [35].

Одним из способов задания множеств являются характеристические функции. Пусть U – некоторое универсальное множество. Характеристическая

функция множества $A \subseteq U$ – это функция $\mu_A : x \to \{0,1\}$, значения которой (0 или 1) указывают на то, является $x \in U$ элементом множества А или нет:

$$\mu_A = \begin{cases} 0, x \in A; \\ 1, x \notin A. \end{cases}$$

Нечеткие множества являются обобщением обычных множеств, для случая, когда характеристическая функция может принимать любые значения из отрезка [0, 1]. В теории нечетких множеств данная характеристическая функция называется функцией принадлежности, а ее значение $\mu_A(x)$ – степень принадлежности элемента x нечеткому множеству А.

Функции принадлежности задавались треугольными и трапециевидными формами кривых. Треугольная функция определяется тройкой чисел (a, b, c), и ее значение в точке x вычисляется согласно выражению

$$\mu(x) = \begin{cases} 1 - \dfrac{b - x}{b - a}, a \le x \le b; \\ 1 - \dfrac{x - c}{c - b}, b \le x \le c; \\ 0, в\_остальных\_случаях. \end{cases}$$

Для задания трапецеидальной функции принадлежности используется четверка чисел (a, b, c, d).

$$\mu(x) = \begin{cases} 1 - \dfrac{b - x}{b - a}, a \le x \le b; \\ 1, b \le x \le c; \\ 1 - \dfrac{x - c}{d - c}, c \le x \le d; \\ 0, в\_остальных\_случаях. \end{cases}$$

Нечеткая переменная определяется тройкой $< \alpha, U, \mu >$, где μ – наименование переменной; U – универсальное множество; $\mu = \mu(x)$ – функция принадлежности, определённая на U и характеризующая нашу степень уверенности в том, что x является значением нечеткой переменной.

Лингвистической переменной называется набор $< \beta, \boldsymbol{T}, \boldsymbol{U}, \boldsymbol{G}, \boldsymbol{M} >$, где β – наименование лингвистической переменной; $\boldsymbol{T}$ – множество ее значений, представляющих собой наименования нечетких переменных; $\boldsymbol{G}$ – синтаксическая процедура, позволяющая оперировать элементами $\boldsymbol{T}$, в частности генерировать новые значения; $\boldsymbol{M}$ – семантическая процедура, позволяет превратить каждое значение лингвистической переменной, образуемое процедурой $\boldsymbol{G}$, в нечеткую переменную.

Нечетким логическим выводом называется аппроксимация зависимости $\boldsymbol{Y} = \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ выходной лингвистической переменной от входных лингвистических переменных и получение заключения в виде нечеткого множества с использованием базы знаний, содержащей правила в виде “Если…, то…” [17, 18].

В качестве признаков мы взяли выявленные в процессе анализа вредоносных программ свойства, рассмотренные во второй главе, сформулировали набор нечетких логических переменных и построили графики функций принадлежности для заданных нечетких логических переменных:

1. Источник, в котором располагается программа: не доверенный, неизвестный, известный. Под источником понимается объект или ресурс, из которого была получена программа. Это может быть доверенный, например, встроенный облачный сервис, или посторонний неизвестный ресурс;

2. Количество опасных разрешений: мало, средне, много, очень много. Согласно опыту, полученному в процессе анализа свойств, присущих вредоносным программам, было выявлено, что, как правило, вредоносные программы содержат большое количество разрешений в своем запросе;

3. Наличие подозрительных классов в коде программ: отсутствуют, частичное совпадение, явное совпадение. Классы могут быть задействованы для шифрования, фоновых звонков, перехвата и отправки смс;

4. Наличие в базе сигнатур: отсутствует, присутствует. Сигнатурный метод анализа является эффективным методом защиты;

5. Результат классификации машины опорных векторов: ok, virus;

Выходным значением будет результат работы системы:

6. Результат: безопасная, подозрительная, опасная, очень опасная.

Составленная модель системы поддержки принятия решений на основе нечеткой логики имеет 5 входных знчений, базу правил и одно выходное значение, которое выступает в качестве результата.

Экспертным методом была составлена база правил, необходимая для работы системы с учетом всех возможных вариантов [97].

Таким образом, была реализована модель поддержки принятия решения. Машина опорных векторов позволяет с большим процентом точности выполнить классификацию вредоносных программ и показать свою высокую эффективность по сравнению с классическими методами и нейронными сетями. Модель поддержки принятия решения позволяет внести дополнительный коэффициент точности в результат работы в условиях помех, обладает гибкостью и максимально точно выполняет поставленную задачу в реальных условиях, что окажет положительное влияние на корректность работы системы обнаружения вредоносных программ.

3.7 Описание модели системы обнаружения вредоносных программ

В данном пункте описывается модель системы обнаружения, построенная с применением машины опорных векторов и системы поддержки принятия решений, реализованной на нечеткой логике (рисунок 3.15). Машина опорных векторов обучается и тестируется на выборке, в которой каждому классу составлены векторы поведения программ, затем выполняет классификацию на два класса virus и ok. С помощью системы поддержки принятия решений на основе данных результата работы машины опорных векторов и дополнительных критериев осуществляется вывод окончательного результата в процентном соотношении.

Рисунок 3.15 – Схема модели системы обнаружения вредоносных программ

Система обнаружения вредоносных программ представляет собой объединение двух методов. В качестве входных данных используется разработанная экспериментальная выборка, полученная путем анализа работы системы в целом.

Обучение машины опорных векторов для классификации программ проходит в два этапа. На первом этапе производится обучение на экспериментальной выборке машины опорных векторов, в результате чего она становится способной корректно распределять по классам программы. Обучающая выборка включает в себя список доступных 152 разрешений на запуск и использование ресурсов, необходимых для работы программы, а также выявленных 12 признаков, присущих поведению программ. Таким образом, были составлены векторы поведения программ как вредоносных, так и безопасных. В состав обучающей выборки вошли 67 составленных векторов программ. Тестовая выборка включала в себя 33 вектора программ с внесением изменений с целью усложнить задачу обнаружения вредоносных программ.

На втором этапе формализуются дополнительные признаки и задаются в виде функций принадлежности для системы поддержки принятия решений. В состав функций принадлежности также входит результат классификации машиной опорных векторов. Затем формируется база правил для корректного функционирования системы поддержки принятия решений. На основании всех признаков и результатов работы машины опорных векторов получаем результат, выраженный в процентах.

Алгоритм функционирования системы обнаружения вредоносных программ состоит из следующих шагов:

1) извлечение и формирование вектора признаков из программы,

2) подается вектор признаков программы (вредоносная или безопасная программа) на вход SVM-классификатора,

3) осуществляется классификация машиной опорных векторов на два класса virus и ok,

4) результат классификации и дополнительные признаки, заданные в виде функций, подаются блоку “система поддержки принятия решений”,

5) на основании правил производится анализ результатов,

6) выводится результат в процентах.

Таким образом, модель системы обнаружения вредоносных программ имеет в своем составе два блока: машина опорных векторов, которая осуществляет классификацию с высокой эффективностью, и система поддержки принятия решений на основе нечеткой логики, которая позволяет повысить точность классификации машиной опорных векторов с учетом помех и дать точный результат, выраженный в процентах. Таким образом, можно осуществлять обнаружение вредоносных программ при их отсутствии в базе сигнатур, опираясь на поведение, присущее вредоносным и безопасным программам.

3.8 Моделирование работы системы обнаружения вредоносных программ

Даны векторы состояний программ: virus и ok, также заданы дополнительные условия – функции принадлежности. Необходимо выполнить классификацию заданного вектора машиной опорных векторов, а также системой поддержки принятия решений выявить процент опасности программы с учетом результата классификации машины опорных векторов и дополнительных условий, заданных в виде функций принадлежности при различных условиях и с учетом помех.

Обучим машину опорных векторов на имеющейся выборке и рассмотрим работу модели системы обнаружения вредоносных программ в различных условиях с учетом помех, чтобы оценить эффективность и качество работы системы.

Рассмотрим простой пример, характеризующий идеальные условия работы: машине опорных векторов подается вектор, на 100 % характеризующий поведение вредоносной программы, системе поддержки принятия решений задаются условия в виде функций принадлежности, явно характеризующие поведение вредоносной программы: источник не доверенный, много разрешений, явное совпадение в классе, присутствие в базе сигнатур.

Подадим обученной машине опорных векторов вектор, характеризующий поведение на 100 % вредоносной программы, заданный в бинарной форме:
1000000000001000000000000000000000000011010000110000000000000110100100
0000001010000000010011010100000000000000000110000001000000000010000000
00001011110100000000000 – virus.

Окно ввода значений вектора представлено на рисунке 3.16: столбец справа содержит название переменной, а слева – вводимое пользователем значение этой переменной.

Окно ввода вектора содержит 163 ячейки для каждого признака. Вектор будет вводиться в окне ввода и подаваться обученной машине опорных векторов для классификации.

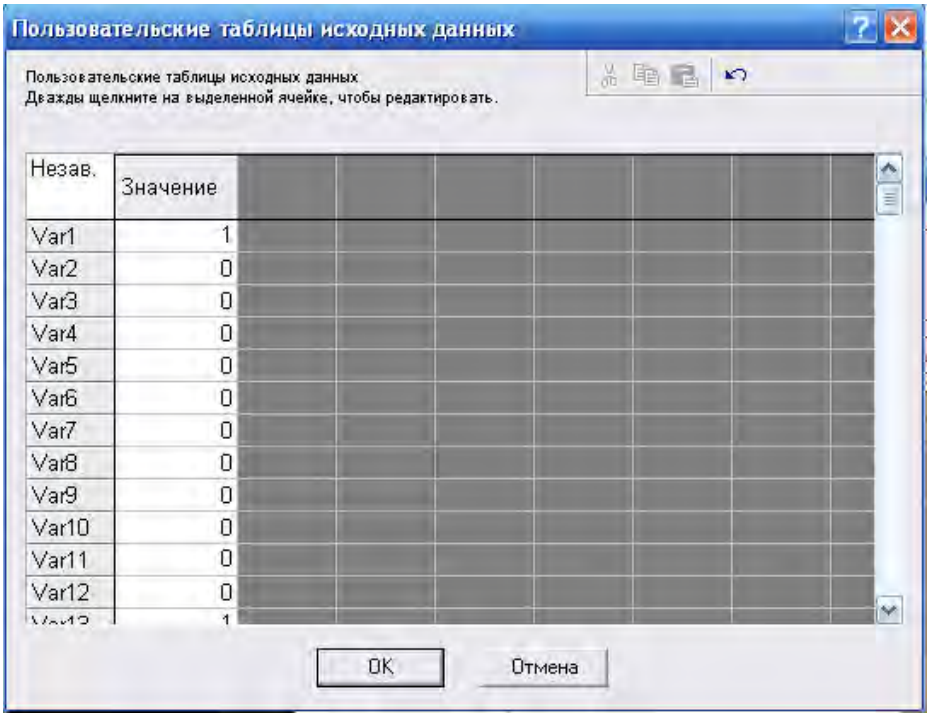


Рисунок 3.16 – Ввод параметров вектора, характеризующего поведение программы

На следующем шаге обученная машина опорных векторов выполняет классификацию и предоставляет результат работы в виде ok или virus в виде таблицы на рисунке 3.17.

Следующий этап моделирования – система поддержки принятия решений, одним из входных значений которой является результат работы обученной машины опорных векторов, а также дополнительные логические переменные, описанные выше. На ее входы подаются следующие значения: источник не доверенный, много опасных разрешений, явное совпадение в классе, присутствие в базе сигнатур и полученный результат классификации машины опорных векторов: virus.

Var139	0,000000
Var140	1,000000
Var141	0,000000
Var142	0,000000
Var143	0,000000
Var144	0,000000
Var145	0,000000
Var146	0,000000
Var147	0,000000
Var148	0,000000
Var149	0,000000
Var150	0,000000
Var151	0,000000
Var152	1,000000
Var153	0,000000
Var154	1,000000
Var155	1,000000
Var156	1,000000
Var157	1,000000
Var158	0,000000
Var159	1,000000
Var160	0,000000
Var161	0,000000
Var162	0,000000
Var163	virus

Рисунок 3.17 – Результат классификации вектора, на 100 % характеризующего поведение вредоносной программы

Введем полученные условия системе поддержки принятия решений.

В результате работы системы обнаружения было выявлено, что программа является на 83 % вредоносной.

Внесем намеренные помехи, то есть машине опорных векторов будет подаваться вектор, характеризующий поведение на 70 % вредоносной программы, и системе поддержки принятия решений условия, в меньшей степени характеризующие поведение вредоносной программы: источник неизвестный, среднее количество разрешений, частичное совпадение в классе, присутствие в базе сигнатур.

Подадим обученной машине опорных векторов вектор, характеризующий поведение на 70 % вредоносной программы, заданный в бинарной форме:

10000000000010000000000000000000000011010000110000000000000110100100
00000010100000000100110101000000000000000001000000000000000000000000000
00000000000000000000 – virus.

При условии 70 % помех машина опорных векторов классифицировала вектор как virus.

Далее также системе поддержки принятия решений введем значения с учетом классификации машины опорных векторов: источник неизвестный, среднее количество разрешений, частичное совпадение в классе, присутствие в базе сигнатур и полученный результат классификации машины опорных векторов: virus.

В результате работы системы обнаружения было установлено, что программа является на 58 % вредоносной.

Пусть пользователь получает программу из определенного неизвестного интернет-ресурса. Вредоносная программа, которую он переносит на устройство, новая и отсутствует в базе сигнатур, но в ней присутствует много опасных разрешений и частично опасные классы. Согласно данным условиям протестируем работу системы обнаружения. При этом будет внесено 60 % помех в вектор поведения вредоносной программы. Таким образом получится:
10000000000010000000000000000000000011010000110000000000000110100100
00000010100000000100110101000000000000000001000000000000000000000000000
00000000000000000000 – virus.

Машина опорных векторов классифицировала вектор как virus. Зададим вышеизложенные условия для системы поддержки принятия решений.

В результате работы системы было установлено, что программа является вредоносной, то есть опасной, с 50 % уверенностью.

Если учесть ситуацию, когда машина опорных векторов выполнит ошибочную классификацию, а система поддержки принятия решений выполнит дополнительный анализ, то получится следующий результат.

В случае если источник информации будет неизвестный, а количество разрешений мало, у системы обнаружения в числе подозрительных классов будет

частичное совпадение и отсутствие в базе антивирусных сигнатур, то машина опорных векторов классифицирует вредоносную программу как безопасную – ok. Результат классификации 47 %, это означает, что программа, которую проверяли – подозрительная.

Таким образом, эффективность классификации у машины опорных векторов высокая, но в случае сильной “размытости” кластеров на фоне помех могут наблюдаться ошибки в работе. Этим объясняется актуальность использования систем поддержки принятия решений в средствах мобильной связи. Система поддержки принятия решений дополняет работу машины опорных векторов и помогает увеличить точность обнаружения вредоносных программ в условиях помех, а также при появлении ошибок первого и второго рода.

3.9 Выводы по третьей главе

1. Разработана экспериментальная выборка на основе анализа множества вредоносных программ, анализа перечня разрешений и системных вызовов, позволяющая описать поведение двух типов программ: вредоносных и безопасных.

2. На основе исследовательского эксперимента по классификации полученной экспериментальной выборки установлено, что точность работы классических методов классификации (иерархической кластеризации и k-средних) невысокая: ошибки I рода – 26 %, ошибки II рода – 22 %.

3. Показано, что ошибки классификации нейронными сетями присутствуют по всей выборке, но увеличиваются с ростом количества помех, классификация данного вида показала лучший результат по сравнению с классическими методами классификации в условиях с увеличенным количеством помех, но при сильном увеличении помех более чем на 70 % нейронные сети совершают ошибки классификации.

4. Установлено, что машина опорных векторов показала лучшие результаты по сравнению с нейронными сетями и классическими методами. Ошибочно

выполнена классификация 5 типов программ, процент правильно классифицированных составил 72 %. Количество ошибок I рода – 0 %, ошибки II рода – 27 %. В классе virus классификация была выполнена безошибочно и таким образом, что процент правильных составил 100 %.

5. Предложена модель поддержки принятия решения, позволяющая выполнить дополнительную классификацию на основе результата работы машины опорных векторов в условиях помех, она обладает большей гибкостью, что положительно влияет на корректность работы метода в целом.

6. Предложена модель системы обнаружения вредоносных программ на основе двух методов: машины опорных векторов, которая осуществляет классификацию с высокой эффективностью, и системы поддержки принятия решений на основе нечеткой логики, которая позволяет усилить точность классификации машиной опорных векторов в условиях помех и дать точный результат, выраженный в процентах, позволяющая увеличить эффективность классификации экспериментальной выборки.

7. Выполнено моделирование работы системы обнаружения вредоносных программ на основе предложенных методов, что позволило убедиться в высокой эффективности классификации машины опорных векторов и в необходимости использования нечеткой логики в качестве дополнительного метода классификации в условиях сильных помех.

ГЛАВА 4 РЕАЛИЗАЦИЯ ИССЛЕДОВАТЕЛЬСКОГО ПРОТОТИПА МОДЕЛИ СИСТЕМЫ ОБНАРУЖЕНИЯ ВРЕДОНОСНЫХ ПРОГРАММ

4.1 Архитектура исследовательского прототипа модели системы обнаружения вредоносных программ

Разработанный исследовательский прототип модели системы обнаружения вредоносных программ представлен на рисунке 4.1, он содержит следующие блоки:

1) программа представляет собой анализируемую программу, состоящую из архивного файла APK;

2) анализатор выполняет извлечение признаков, а также формирование вектора признаков, который будет подаваться на входы классификатора машины опорных векторов;

3) SVM-классификатор – обученная и протестированная на сформированной обучающей и тестовой выборке машина опорных векторов, выполняющая классификацию вектора признаков на два класса: virus и ok;

4) панель управления является интуитивно понятным пользовательским интерфейсом для работы с системой и содержит перечень всех необходимых инструментов для ввода признаков, генерации статистики и отчетов;

5) обучающая выборка содержит наборы векторов признаков программ, которые состоят из 100 векторов поведения программ: ok и virus, а также необходимы для обучения и тестирования машины опорных векторов;

6) система поддержки принятия решений на основе результата машины опорных векторов, дополнительно заданных признаков, характерных для того или иного типа программ, осуществляет дополнение к результату машины опорных векторов и отображает результат в процентах, а также определяет критерий опасности (безопасная, подозрительная, опасная).

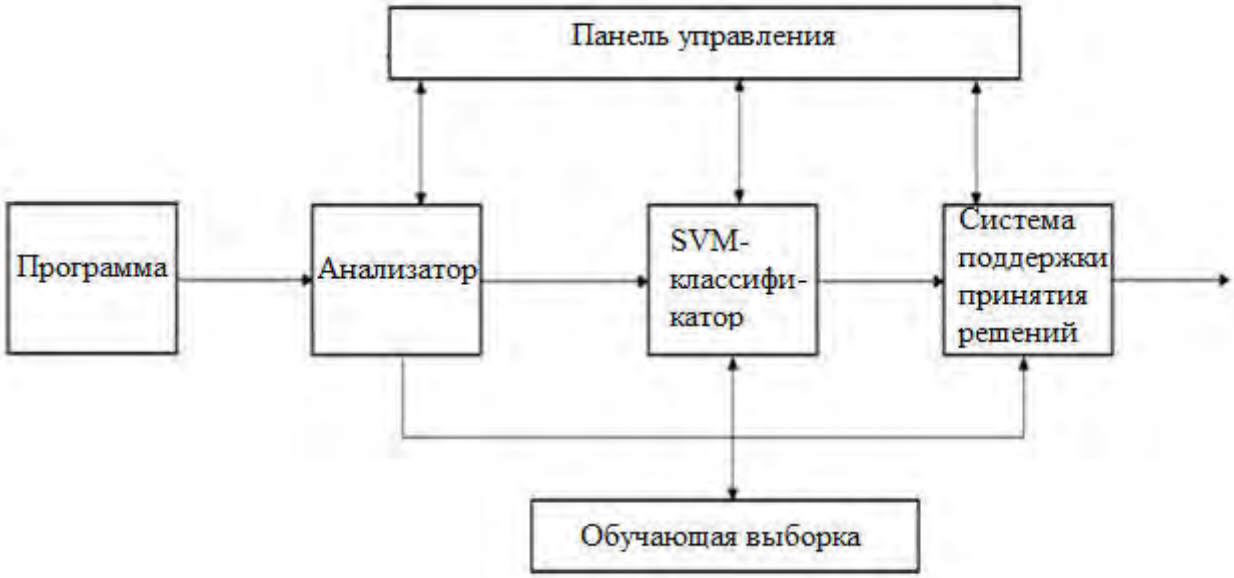


Рисунок 4.1 – Архитектура исследовательского прототипа модели системы обнаружения вредоносных программ

Предложенная архитектура исследовательского прототипа модели системы обнаружения вредоносных программ способна выполнять обнаружение с учетом различных условий и помех, а также повысить эффективность обнаружения вредоносных программ.

4.2 Исследовательский прототип модели системы обнаружения вредоносных программ

На основе проделанного в третьей главе эксперимента был разработан исследовательский прототип модели системы обнаружения вредоносных программ, реализующий предложенные методы: машину опорных векторов и систему поддержки принятия решений на основе нечеткой логики по алгоритму Мамдани. Исследовательский прототип позволяет выполнить анализ и оценку эффективности работы методов при различном уровне помех. Он реализован в виде программы на языке C++. Листинг реализованных в программе алгоритмов представлен в приложении Б.

В программе реализованы два метода: машина опорных векторов и нечеткая логика с алгоритмом Мамдани. Блок-схема работы машины опорных векторов представлена на рисунке 4.2.

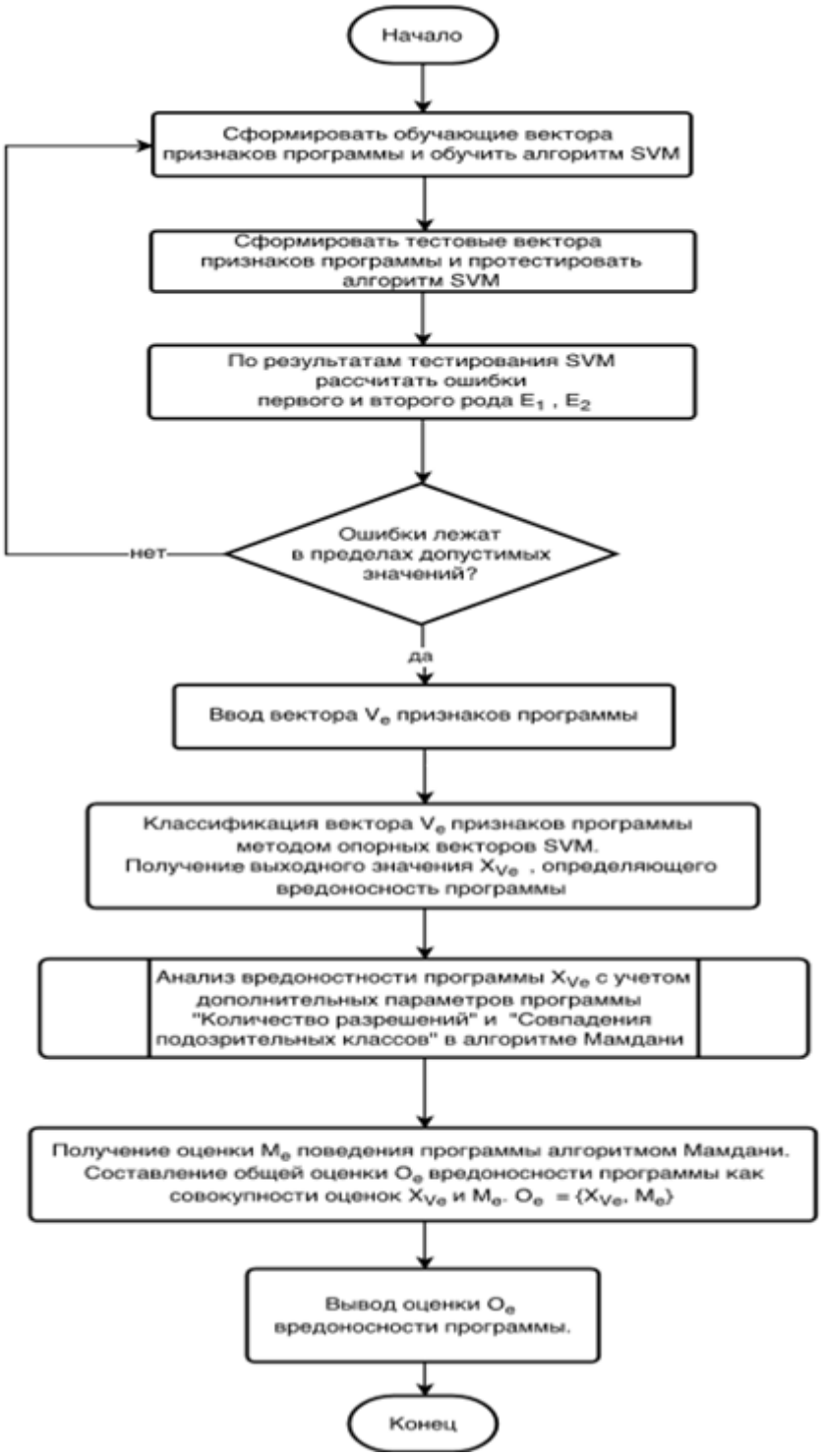


Рисунок 4.2 – Блок-схема работы машины опорных векторов

На начальном этапе машина опорных векторов обучается на обучающей выборке и тестируется на тестовой. Выполняется оценка качества работы

обученной машины опорных векторов при помощи расчета ошибок первого и второго рода. При удовлетворительном результате на входы машины опорных векторов подаются векторы признаков программ для классификации, далее значения передаются на входы нечеткой логики, реализованной на алгоритме Мамдани. Блок-схема работы нечеткой логики на алгоритме Мамдани представлена на рисунке 4.3.

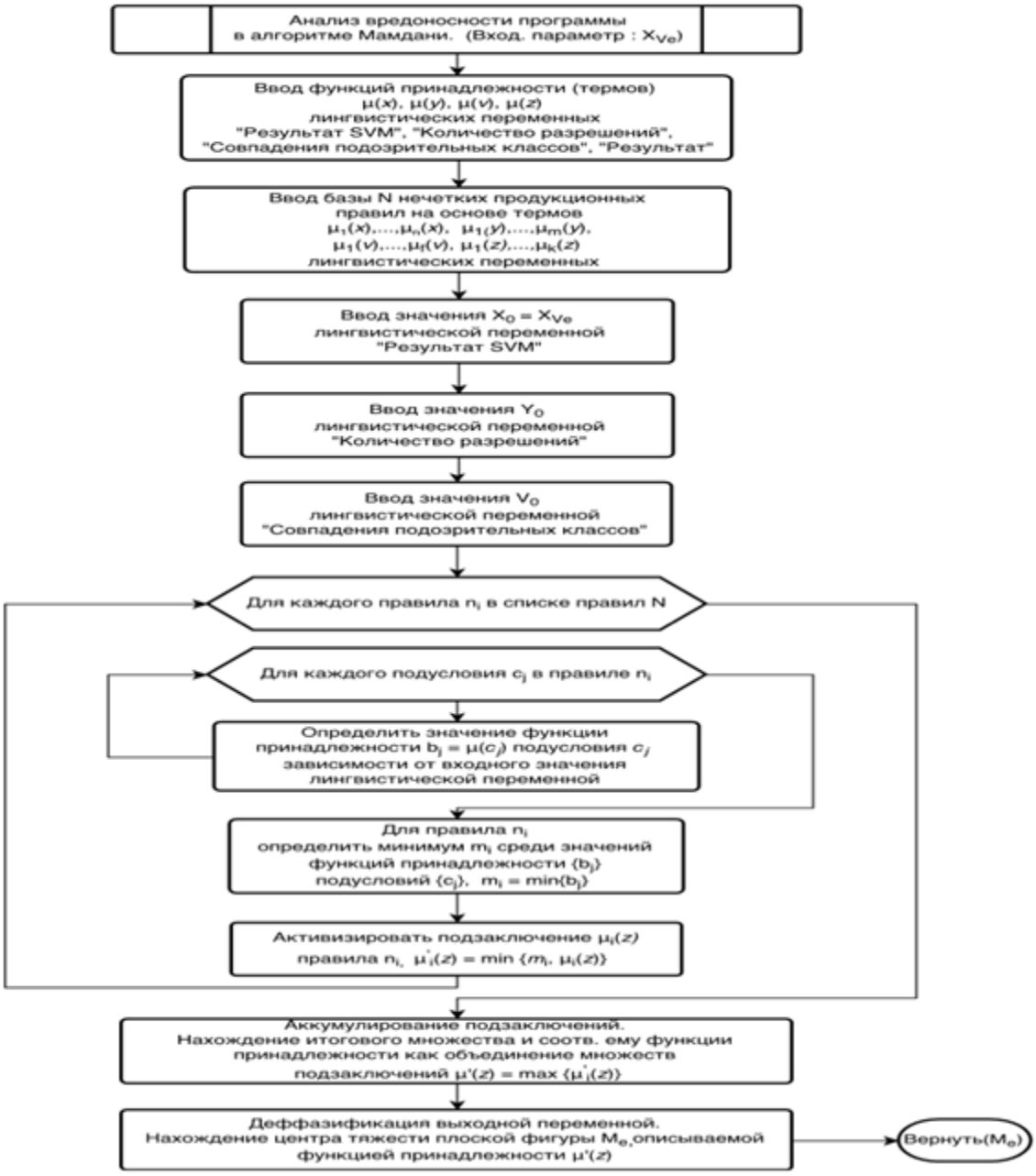


Рисунок 4.3 – Блок-схема работы нечеткой логики

Одной из трех функций принадлежности аппарата нечеткой логики является результат работы машины опорных векторов, а две другие – количество

разрешений и совпадения опасных классов на основе составленных правил – выполняют дополнительный анализ, затем аппарат нечеткой логики выдает результат в процентах и в виде трех критериев: безопасная, подозрительная и опасная.

Программа представляет собой окно с тремя вкладками: главная (рисунок 4.4), статистика (рисунок 4.10) и журнал (рисунок 4.16).

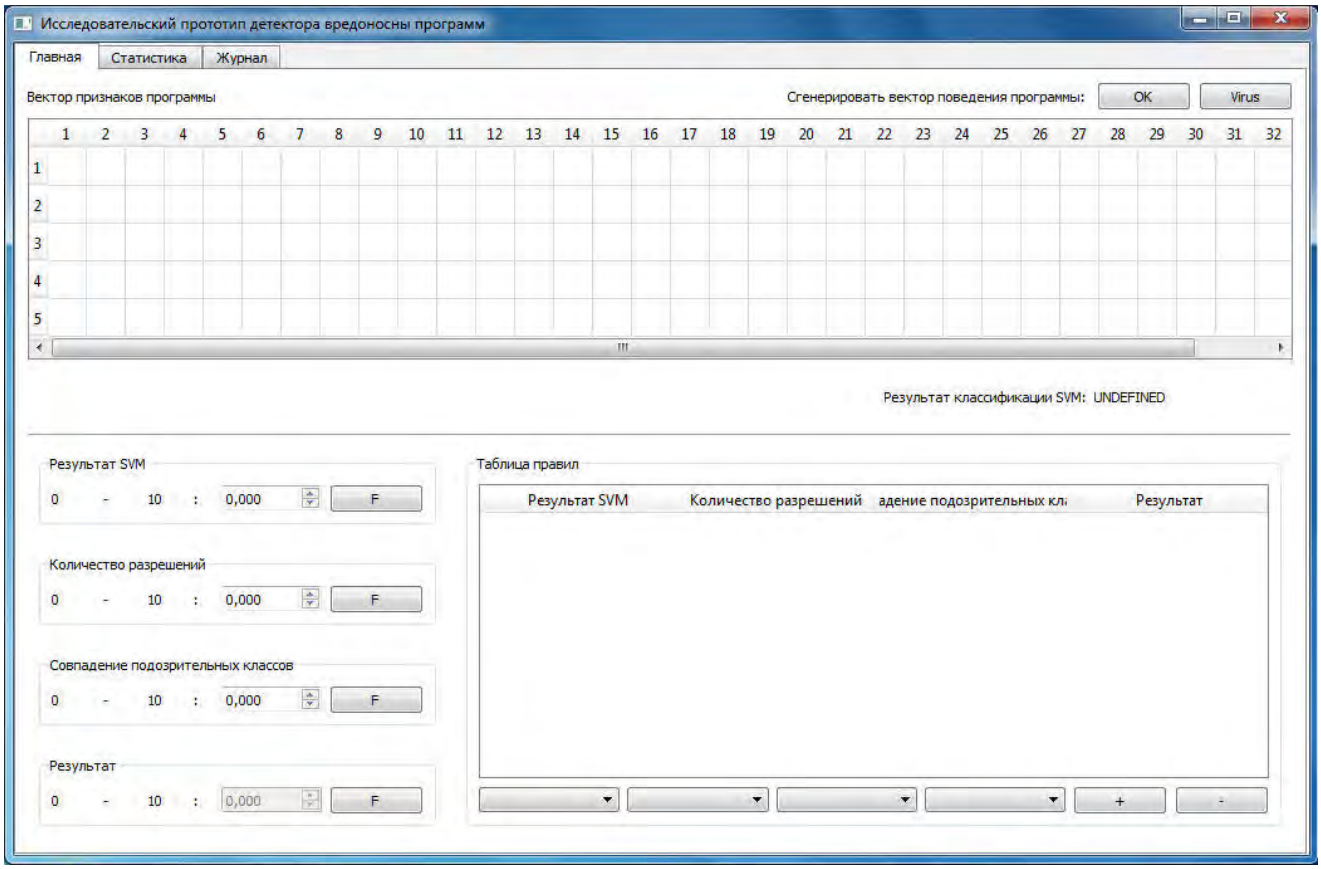


Рисунок 4.4 – Главное окно программы

В окне “Главная” содержится таблица для ввода значений векторов (рисунок 4.5), характеризующих поведение той или иной программы, что позволяет внести необходимое количество помех в рассматриваемый вектор. Также можно автоматически сгенерировать необходимый вектор программы: ok или virus, при этом машина опорных векторов не будет иметь представления о том, какой тип программы сгенерирован. Она будет выполнять классификацию и выводить результат.

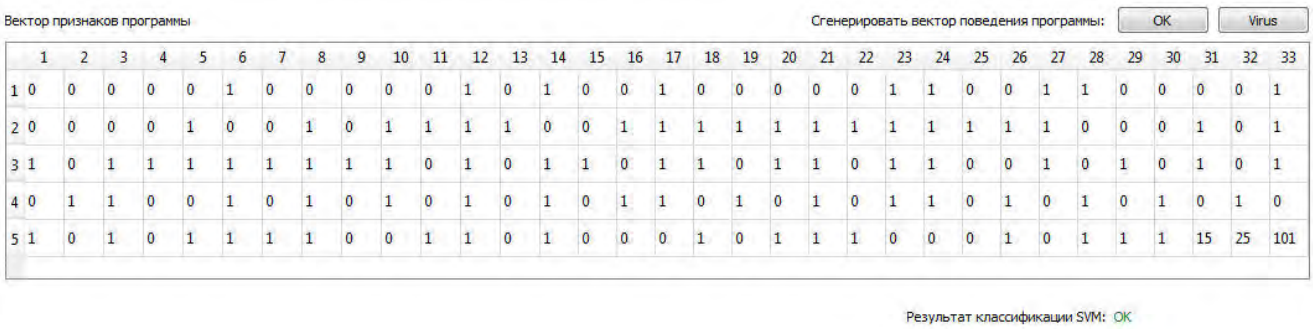


Рисунок 4.5 – Поле ввода вектора и кнопки для генерирования необходимых векторов

На рисунке 4.6 представлен результат классификации уже обученной машиной опорных векторов.

Рисунок 4.6 – Результат классификации машины опорных векторов

В нижней области окна “Главная” находятся все инструменты для ввода параметров (функций принадлежности и их значений) нечеткой логики (рисунок 4.7).

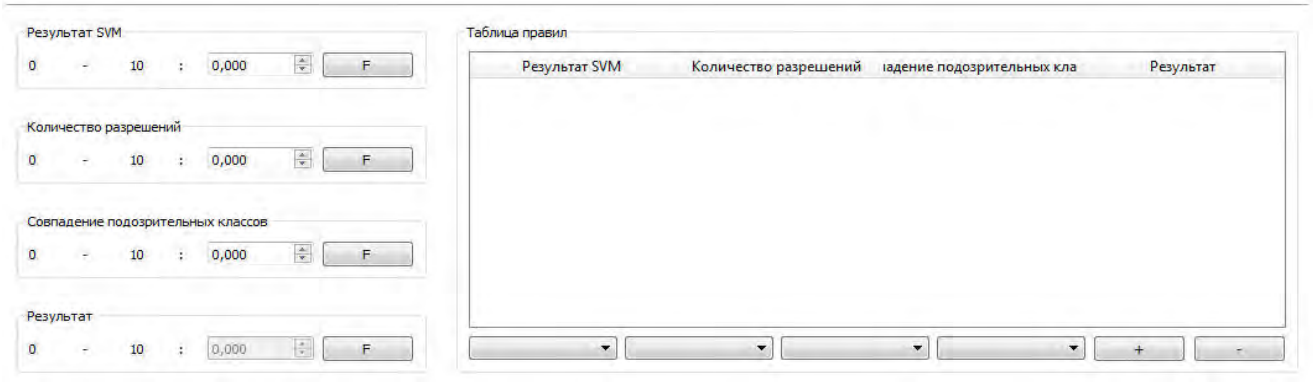


Рисунок 4.7 – Окно ввода нечеткой логики

В данном окне задаются функции принадлежности для входных и выходных значений, правила, а также задаются условия, при которых нечеткая логика будет выполнять свою работу. При нажатии на кнопку “F” появляется окно для ввода

функций принадлежности. Окно ввода одной из функций принадлежности “Результат SVM” представлен на рисунке 4.8.

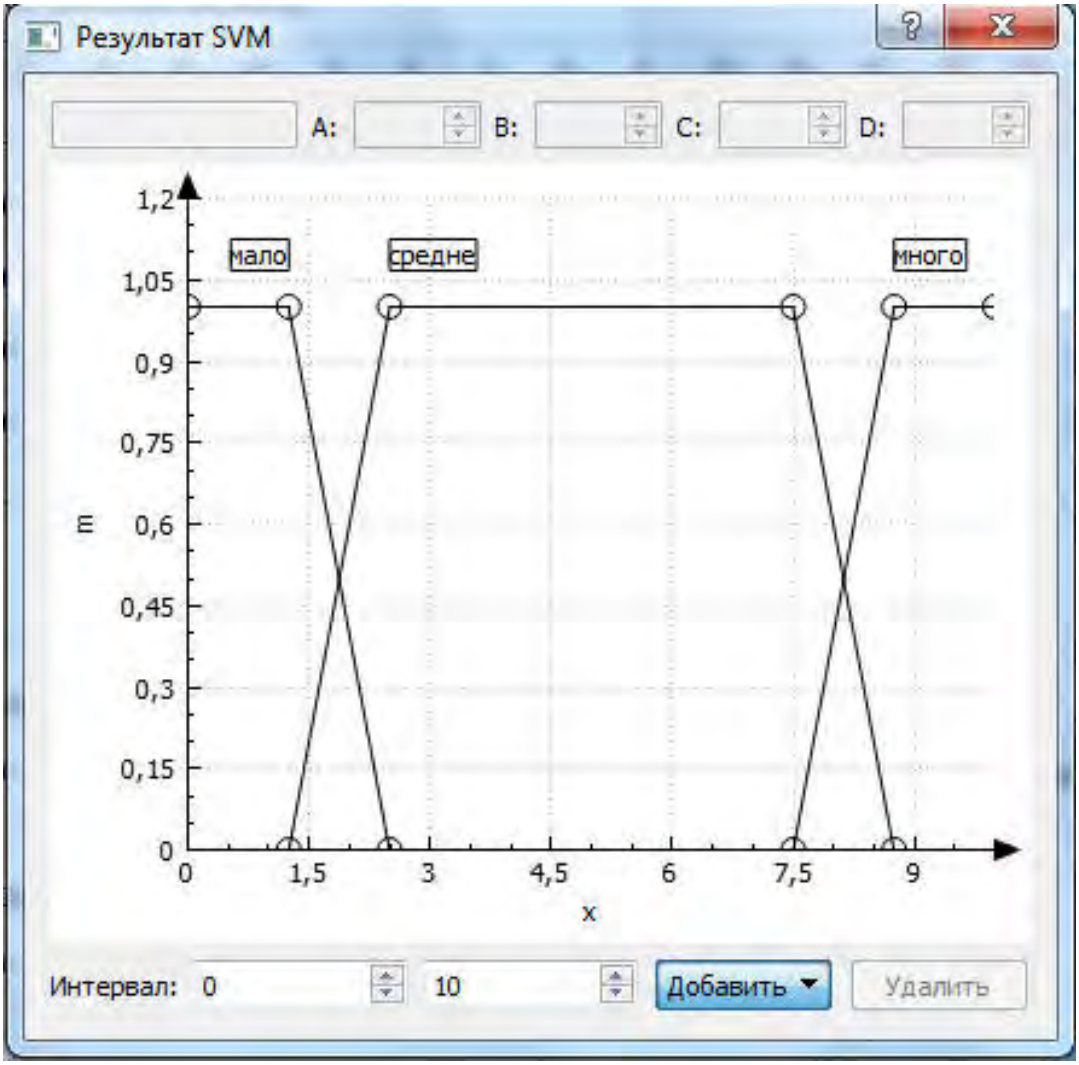


Рисунок 4.8 – Окно ввода функций принадлежности

В данном окне задаются интервал, названия и типы графиков.

После ввода всех функций принадлежности необходимо ввести правила в таблицу правил (рисунок 4.9).

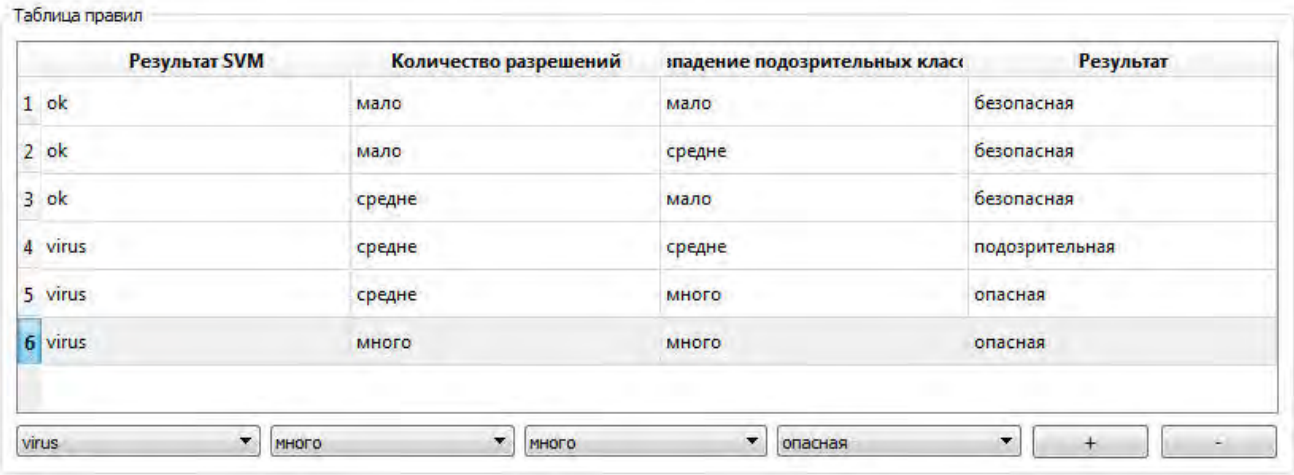


Рисунок 4.9 – Таблица ввода правил

Во вкладке “Статистика” содержится вся информации о проделанных экспериментах, представляющая собой таблицу с двумя столбцами: эталонная модель и результат работы машины опорных векторов.

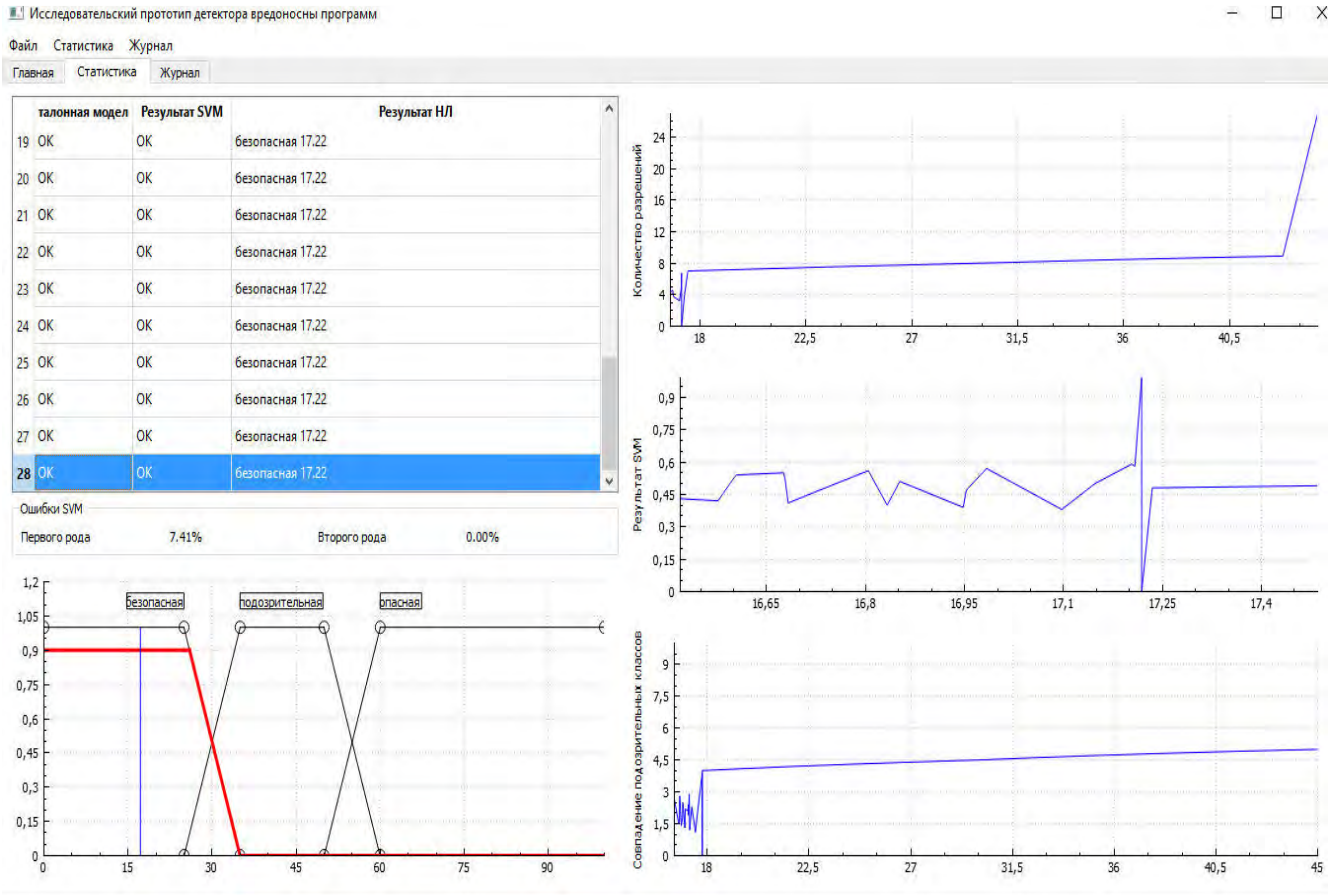


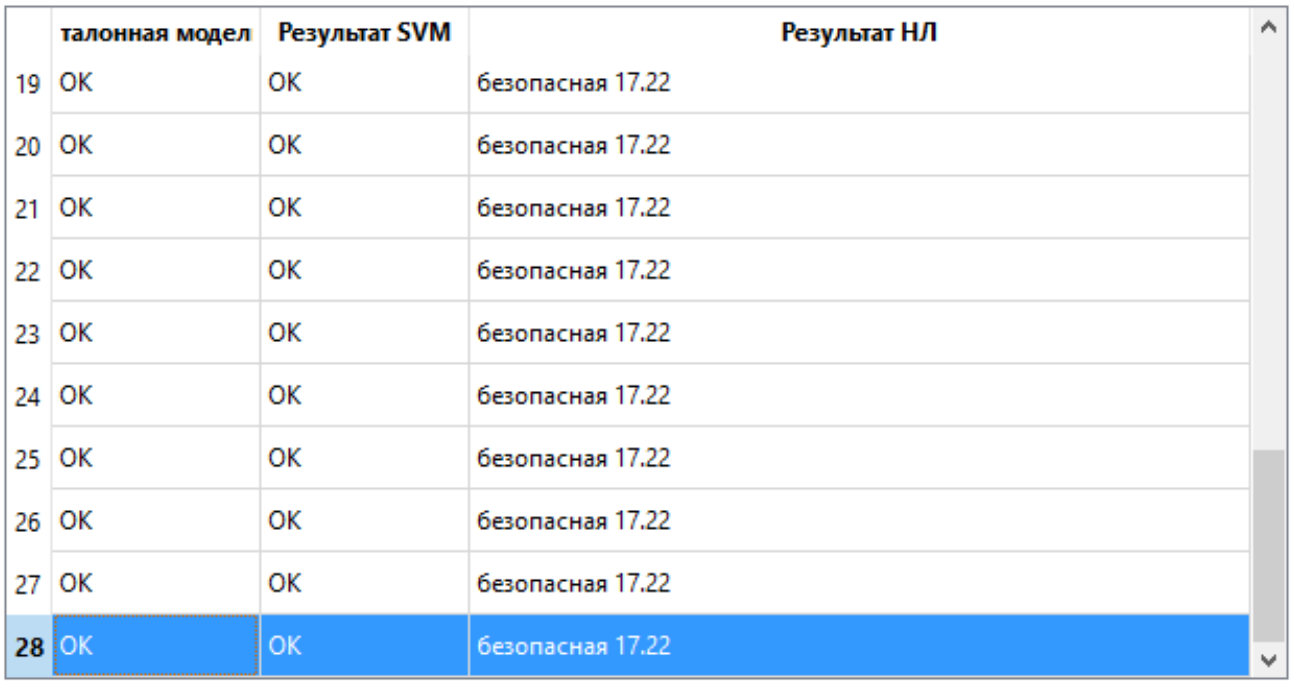
Рисунок 4.10 – Вкладка окна “Статистика”

В данной вкладке по результатам эксперимента выполняется подсчет ошибок первого и второго рода. Вкладка содержит таблицу с тремя колонками (рисунок 4.11):

1) эталонная модель – этот тот результат, который ожидается;

2) SVM – фактический результат классификации машины опорных векторов;

3) результат НЛ – фактический результат системы поддержки принятия решений на основе нечеткой логики по алгоритму Мамдани.



	талонная модел	Результат SVM	Результат НЛ
19	OK	OK	безопасная 17.22
20	OK	OK	безопасная 17.22
21	OK	OK	безопасная 17.22
22	OK	OK	безопасная 17.22
23	OK	OK	безопасная 17.22
24	OK	OK	безопасная 17.22
25	OK	OK	безопасная 17.22
26	OK	OK	безопасная 17.22
27	OK	OK	безопасная 17.22
28	OK	OK	безопасная 17.22

Рисунок 4.11 – Таблица окна “Статистика”

Во вкладке “Статистика” отображается результат работы всей системы в целом, в частности: результат выходной функции принадлежности системы поддержки принятия решений на основе нечеткой логики, согласно которой можно проследить за ходом исследовательского эксперимента с учетом работы нечеткой логики и воздействия всех правил и функций принадлежности. В данном окне формируется статистика по результатам проделанного эксперимента, согласно которой рассчитываются ошибки первого и второго рода,

осуществляется построение зависимостей. На рисунке 4.12 представлена зависимость в виде результата выходной функции нечеткой логики.

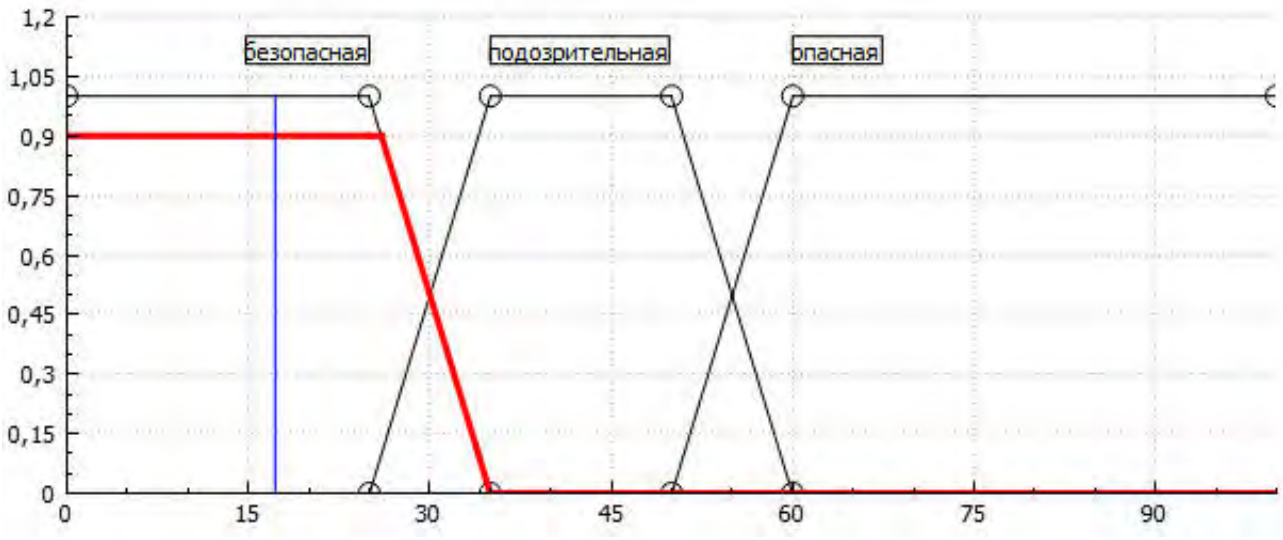


Рисунок 4.12 – Выходная функция принадлежности

Также в окне “Статистика” отображаются три зависимости:

1. Зависимость количества разрешений от результата (рисунок 4.13);

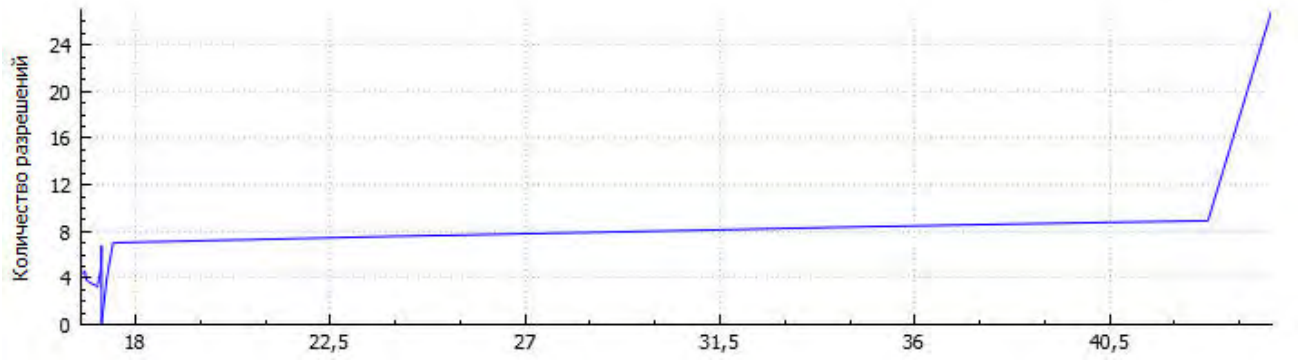


Рисунок 4.13 – Зависимость 1

2. Зависимость результата SVM от результата нечеткой логики (рисунок 4.14);

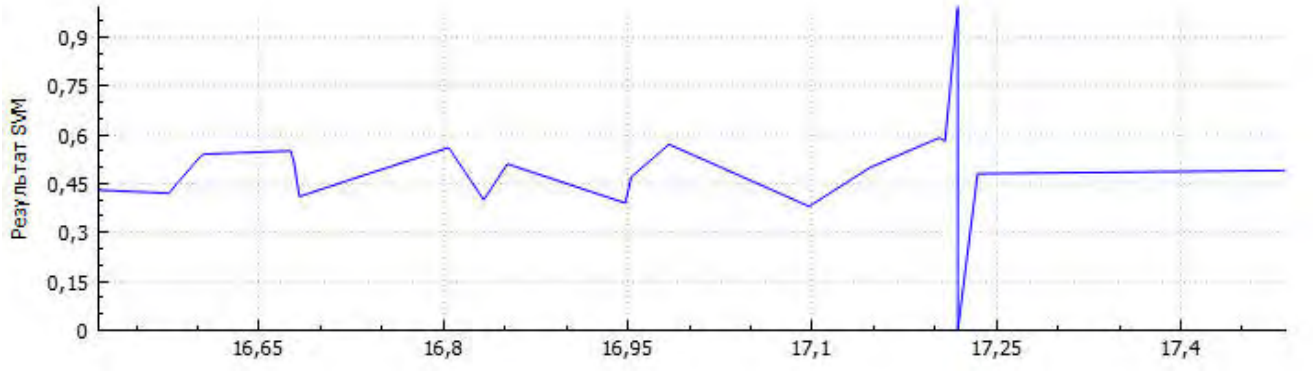


Рисунок 4.14 – Зависимость 2

3. Зависимость совпадения подозрительных классов от результата (рисунок 4.15).

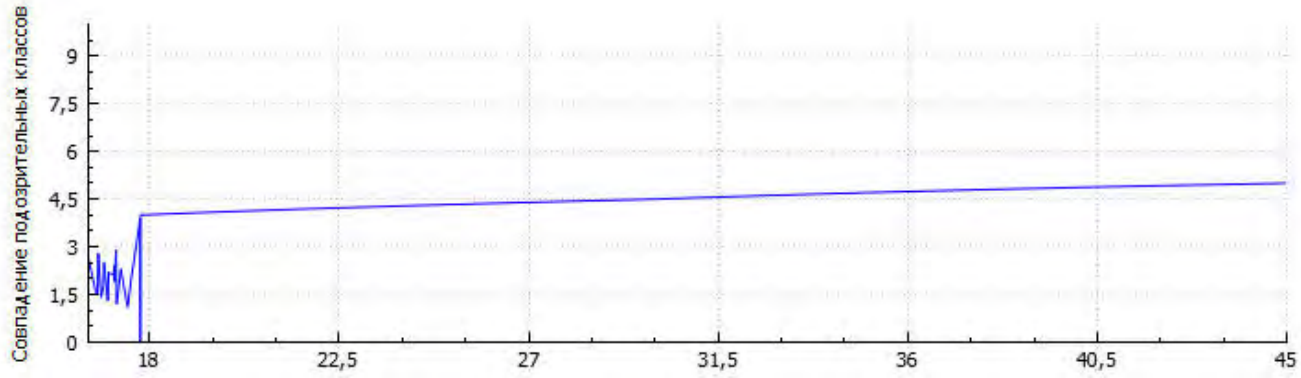


Рисунок 4.15 – Зависимость 3

Выполняя анализ зависимостей, можно сделать выводы о воздействии входных функций принадлежности на результат работы нечеткой логики.

Во вкладке “Журнал” ведется статистика всех проделанных действий, связанных с работой программы (рисунок 4.16).

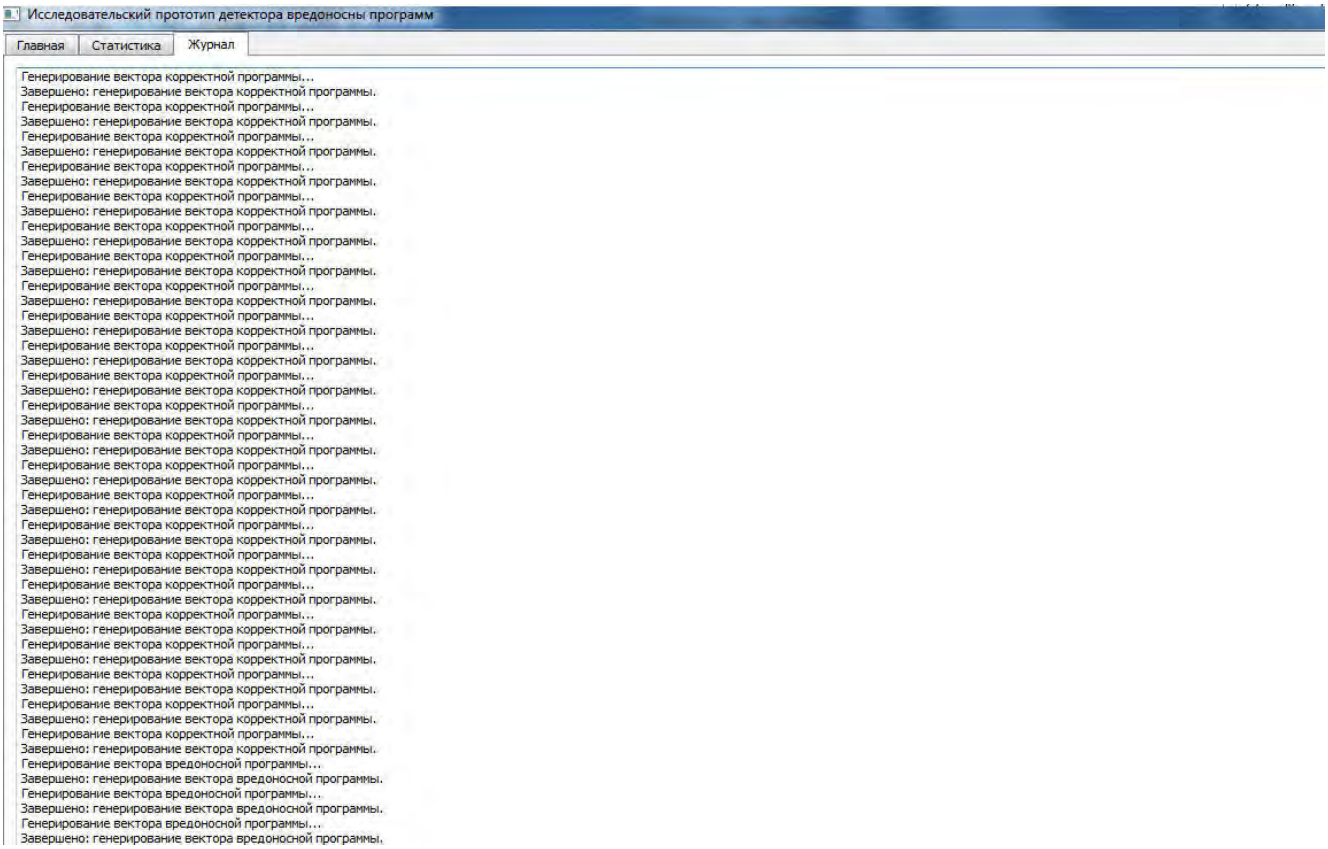


Рисунок 4.16 – Вкладка окна “Журнал”

В разработанной программе имеется возможность загружать значения векторов поведения программ вручную.

Таким образом, была разработана модель исследовательского прототипа системы обнаружения вредоносных программ. Программа обладает достаточным функционалом и набором данных для анализа результатов работы.

4.3 Реализация эксперимента на исследовательском прототипе модели системы обнаружения вредоносных программ

С целью проверки эффективности работы разработанной модели и сравнения с имеющимися антивирусными решениями по защите от вредоносных программ, проделаем эксперимент, а также выполним анализ полученных данных с учетом различных уровней помех.

Для выполнения эксперимента были сгенерированы 100 векторов признаков: 40 значений ok и 60 значения virus.

В окне ввода данных (рисунок 4.17) генерируются векторы признаков в произвольной форме и меняются входные функции принадлежности в нечеткой логике.

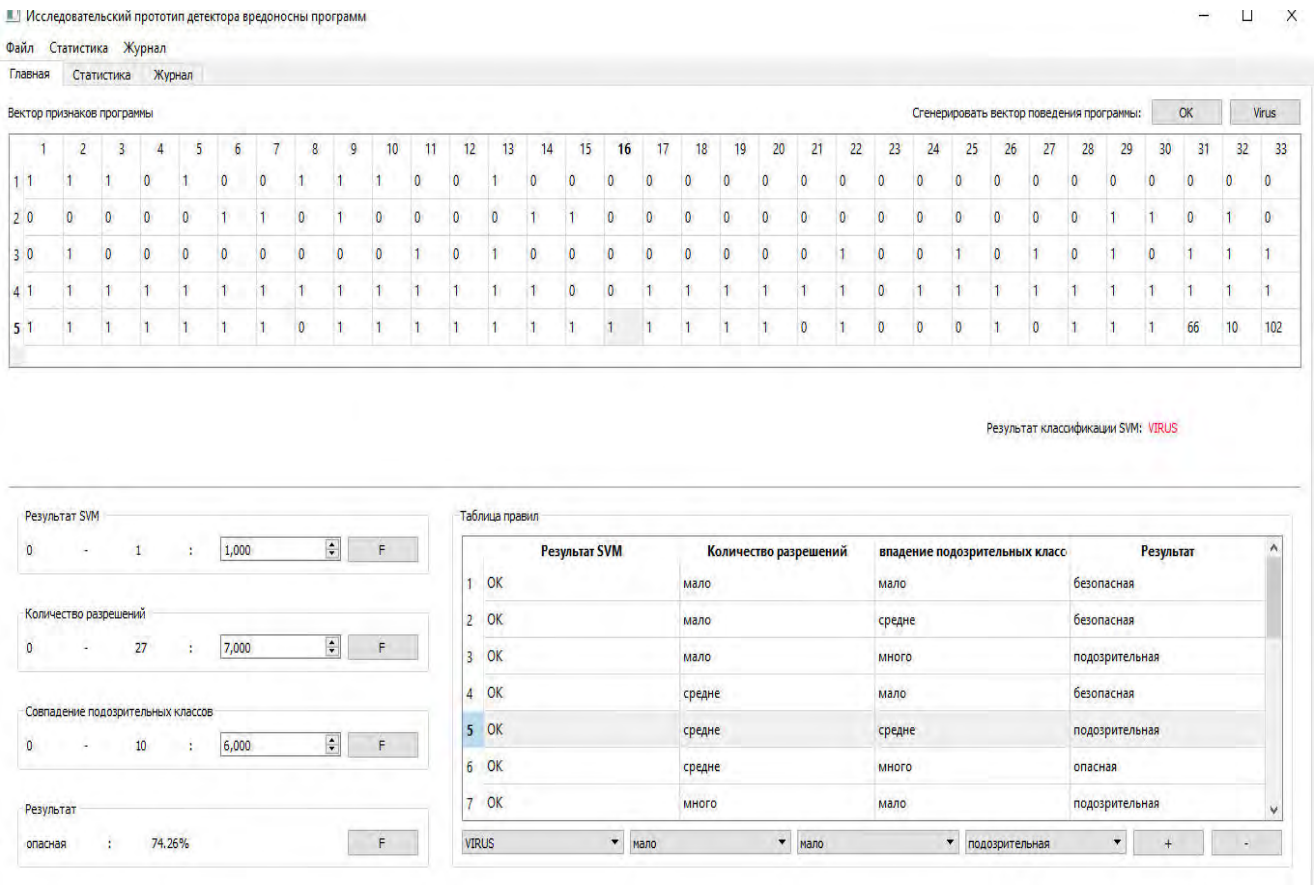


Рисунок 4.17 – Окно ввода данных

Для генерации векторов признаков используем кнопки ok и virus в верхнем правом углу. Для ввода нечетких входных функций принадлежности будем использовать поля ввода в нижнем левом углу. По результатам проделанного эксперимента программа собрала статистику, посчитала ошибки первого и второго рода и составила зависимости (рисунок 4.18).

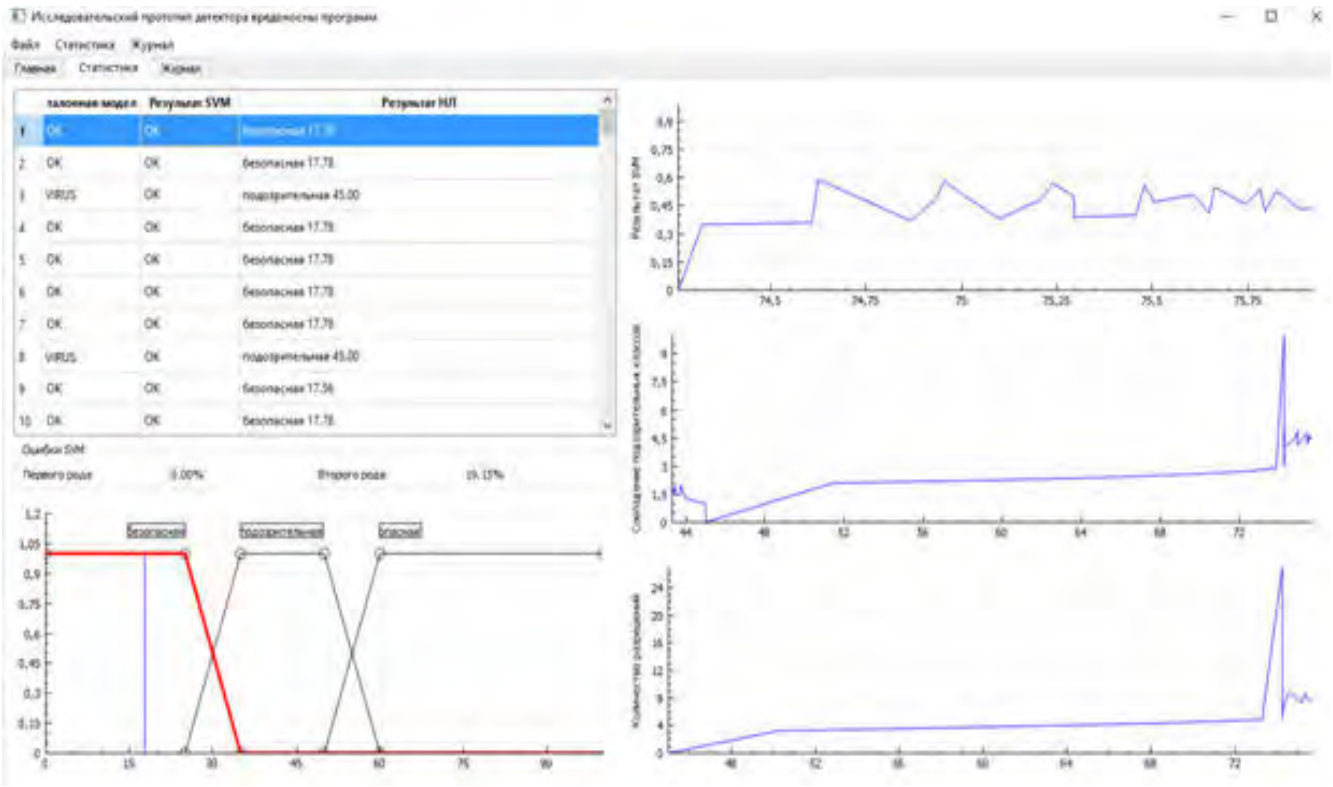


Рисунок 4.18 – Статистика проделанного эксперимента

Для удобства просмотра статистики значения составлена таблица 4.1.

Таблица 4.1 – Результаты проделанной работы

№	Эталонные значения	Результат SVM	Входные функции принадлежности нечеткой логики
1	2	3	4
1	ok	ok	безопасная 17,78
2	ok	ok	безопасная 17,78
3	virus	ok	подозрительная 45,00
4	ok	ok	безопасная 17,78
5	ok	ok	безопасная 17,78
6	ok	ok	безопасная 17,78
7	ok	ok	безопасная 17,78
8	virus	ok	подозрительная 45,00

Продолжение таблицы 4.1

1	2	3	4
9	ok	ok	безопасная 17,56
10	ok	ok	безопасная 17,78
11	ok	ok	безопасная 17,78
12	ok	ok	безопасная 17,78
13	virus	ok	подозрительная 45,00
14	ok	ok	подозрительная 30,04
15	ok	ok	подозрительная 30,04
16	ok	ok	подозрительная 30,04
17	virus	ok	опасная 63,25
18	ok	ok	подозрительная 30,04
19	ok	ok	подозрительная 30,04
20	ok	ok	безопасная 17,78
21	ok	ok	безопасная 17,78
22	ok	ok	безопасная 17,78
23	ok	ok	безопасная 17,78
24	ok	ok	безопасная 17,78
25	ok	ok	безопасная 17,78
26	ok	ok	безопасная 17,78
27	ok	ok	безопасная 17,78
28	ok	ok	безопасная 17,78
29	virus	ok	подозрительная 45,00
30	virus	ok	подозрительная 45,00
31	ok	ok	опасная 63,25
32	ok	ok	безопасная 17,56
33	ok	ok	подозрительная 43,75
34	ok	ok	опасная 74,83

Продолжение таблицы 4.1

1	2	3	4
35	ok	ok	опасная 74,83
36	ok	ok	безопасная 17,78
37	ok	ok	безопасная 17,78
38	ok	ok	безопасная 17,78
39	ok	ok	безопасная 17,78
40	ok	ok	безопасная 17,78
41	virus	ok	подозрительная 45,00
42	virus	ok	подозрительная 45,00
43	ok	ok	безопасная 17,78
44	ok	ok	безопасная 17,78
45	ok	ok	безопасная 17,78
46	ok	ok	безопасная 17,78
47	ok	ok	безопасная 17,78
48	virus	virus	подозрительная 45,00
49	virus	virus	подозрительная 45,00
50	virus	virus	подозрительная 45,00
51	virus	virus	опасная 63,25
52	virus	virus	опасная 63,25
53	virus	virus	опасная 74,26
54	virus	virus	опасная 74,26
55	virus	virus	опасная 74,26
56	virus	virus	опасная 74,26
57	virus	virus	опасная 74,83
58	virus	virus	опасная 74,26
59	virus	virus	опасная 74,26
60	virus	virus	опасная 74,26

Продолжение таблицы 4.1

1	2	3	4
61	virus	virus	опасная 74,26
62	virus	virus	опасная 74,26
63	virus	virus	опасная 74,26
64	virus	virus	опасная 74,26
65	virus	virus	опасная 74,26
66	virus	virus	опасная 74,26
67	virus	virus	опасная 74,26
68	virus	virus	опасная 74,26
69	virus	virus	опасная 63,25
70	virus	virus	опасная 63,25
71	virus	virus	опасная 63,25
72	virus	virus	опасная 63,25
73	virus	virus	опасная 63,25
74	virus	virus	опасная 63,25
75	virus	virus	опасная 63,25
76	virus	virus	опасная 63,25
77	virus	virus	опасная 63,25
78	virus	virus	опасная 63,25
79	virus	virus	опасная 63,25
80	virus	virus	опасная 63,25
81	virus	virus	опасная 63,25
82	virus	virus	опасная 63,25
83	virus	virus	опасная 63,25
84	virus	virus	опасная 63,25
85	virus	virus	опасная 63,25
86	virus	virus	опасная 63,25

Окончание таблицы 4.1

1	2	3	4
87	virus	virus	опасная 63,25
88	virus	virus	опасная 63,25
89	virus	virus	опасная 63,25
90	virus	virus	опасная 63,25
91	virus	virus	опасная 63,25
92	virus	virus	опасная 63,25
93	virus	virus	опасная 63,25
94	virus	virus	опасная 63,25
95	virus	virus	опасная 63,25
96	virus	virus	опасная 63,25
97	virus	virus	опасная 63,25
98	virus	virus	опасная 74,26
99	virus	virus	опасная 74,26
100	virus	virus	опасная 74,26

Машина опорных векторов совершила ошибку классификации во множестве вредоносных программ – ошибка первого рода составила 0 %, ошибка второго рода 19,35 %. На участках, где машина опорных векторов выполнила ошибочную классификацию, нечеткая логика показала большой процент вероятности того, что программа является вредоносной (рисунки 4.19–4.21).

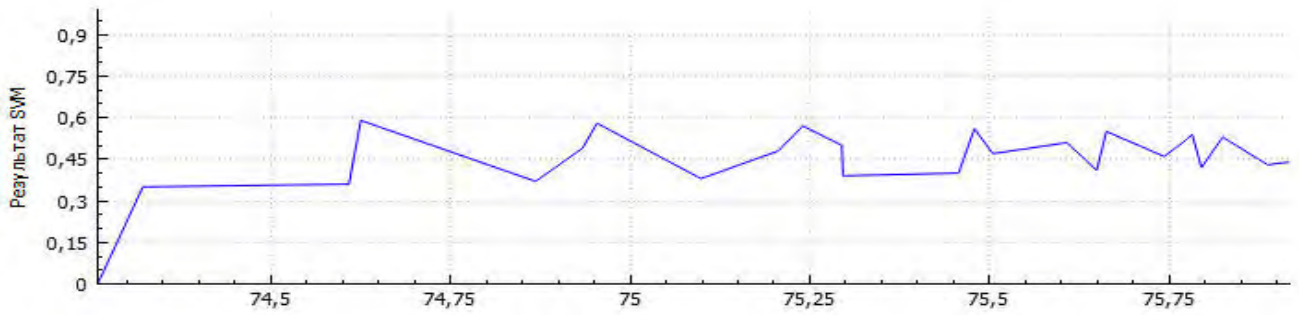


Рисунок 4.19 – Зависимость результата SVM от результата нечеткой логики

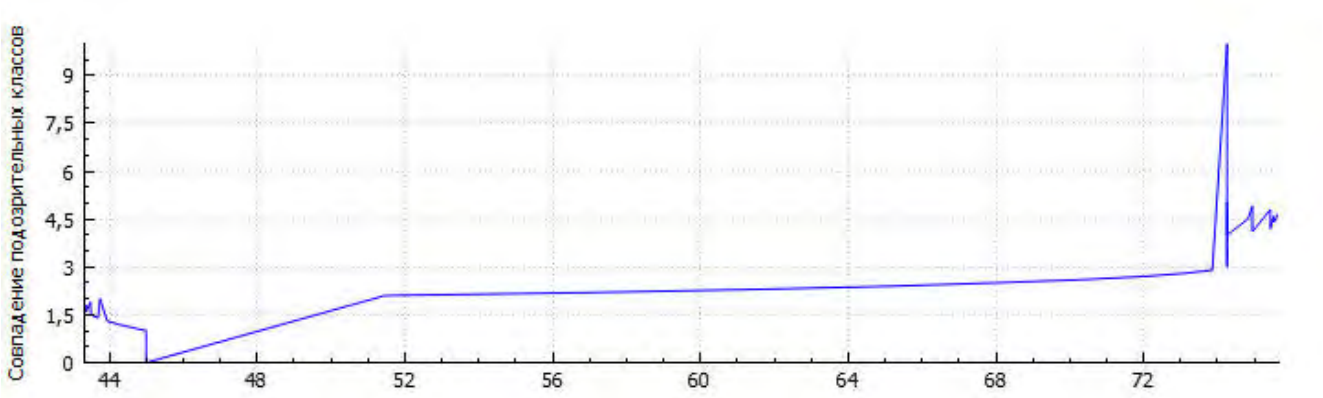


Рисунок 4.20 – Зависимость совпадения подозрительных классов от результата нечеткой логики

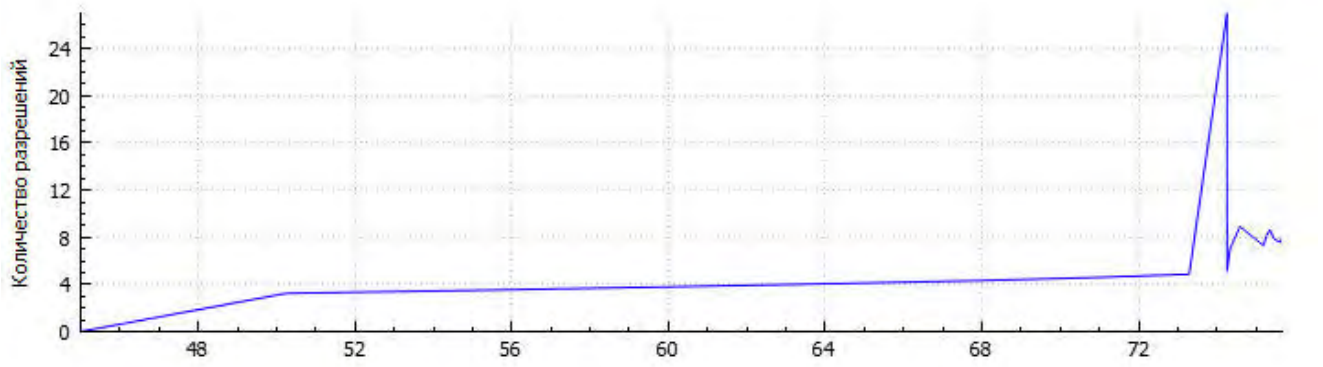


Рисунок 4.21 – Зависимость количества разрешений от результата нечеткой логики

По информации в журнале можно сделать вывод, как изменялись во время эксперимента входные функции принадлежности и какие функции программы были задействованы. Информация из журнала исследовательского прототипа модели системы обнаружения вредоносных программ за текущий эксперимент представлена в приложении В.

Таким образом, в процессе работы машина опорных векторов выполнила классификацию с точностью 80,65 %. Ошибка первого рода составила 0 %, ошибка второго рода 19,35 %. Нечеткая логика выполнила дополнительную классификацию с результатом “подозрительная и опасная программа” для ошибок

второго рода. Нечеткая логика дополняет результат машины опорных векторов и позволяет выполнить корректную классификацию при условии больших помех.

4.4 Оценка эффективности и риска информационной безопасности в ОС для мобильных устройств после внедрения системы обнаружения вредоносных программ

Для сравнения, оценки и анализа эффективности системы обнаружения вредоносных программ введем, обозначим в дополнение к имеющимся на данное время средствам защиты нашу систему обнаружения и определим влияние и риск угроз (таблица 4.2).

Таблица 4.2 – Обозначение и влияние системы обнаружения вредоносных программ.

Обозначение	Название	Влияние концепта на связь
1, 2, 3, 13, 14, 15, 8, 4, 5	Система обнаружения вредоносных программ	Слабо
6, 7, 9, 10		Средне
11, 12, 16, 17, 18, 19, 20, 21		Сильно

Составим нечеткую когнитивную карту с учетом деятельности системы обнаружения вредоносных программ. Определим базу правил влияния вводимой системы обнаружения на изменение силы связи между концептами, которая в существенной степени зависит от специфики исследуемой области, действующих взаимосвязей и допускает некоторую свободу выбора для использования лингвистических переменных (таблица 4.3).

Таблица 4.3 – База правил механизма нечеткого вывода

Влияние системы обнаружения вредоносных программ на связь	Слабо	Средне	Сильно
Слабо	Слабо	Средне	Сильно
Средне	Слабо	Слабо	Средне
Сильно	Слабо	Слабо	Средне

Нечеткая когнитивная карта с учетом внедренной системы обнаружения вредоносных программ будет выглядеть следующим образом (рисунок 4.22).

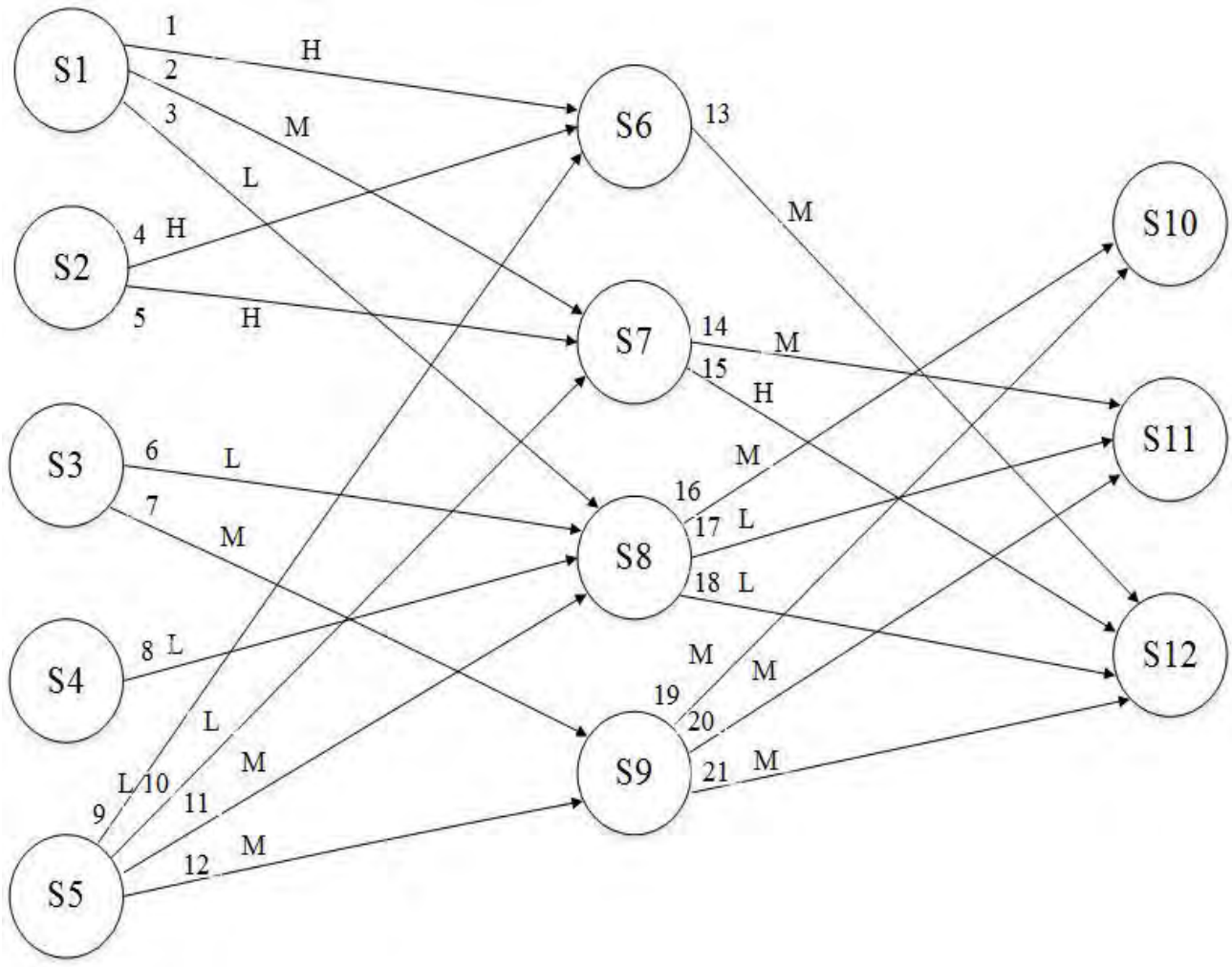


Рисунок 4.22 – Нечеткая когнитивная карта с учетом системы обнаружения вредоносных программ

Составим матрицу достижимости (таблица 4.4). Затем выразим ее экспертным методом (см. пункт 2.4 глава 2) (таблица 4.5).

Таблица 4.4 – Матрица достижимости

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	0	0	0	0	0	H	M	L	0	L	M	M
S2	0	0	0	0	0	H	H	0	0	0	M	H
S3	0	0	0	0	0	0	0	L	M	M	M	M
S4	0	0	0	0	0	0	0	L	0	L	L	L
S5	0	0	0	0	0	L	L	M	M	M	M	M
S6	0	0	0	0	0	0	0	0	0	0	0	M
S7	0	0	0	0	0	0	0	0	0	0	M	H
S8	0	0	0	0	0	0	0	0	0	M	L	L
S9	0	0	0	0	0	0	0	0	0	M	M	M
S10	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	0

Таблица 4.5 – Матрица достижимости, выраженная экспертно

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	0	0	0	0	0	1	0.4	0.2	0	0.2	0.4	0.4
S2	0	0	0	0	0	1	1	0	0	0	0.4	1
S3	0	0	0	0	0	0	0	0.2	0.4	0.4	0.4	0.4
S4	0	0	0	0	0	0	0	0.2	0	0.2	0.2	0.2
S5	0	0	0	0	0	0.2	0.2	0.4	0.4	0.4	0.4	0.4
S6	0	0	0	0	0	0	0	0	0	0	0	0.4
S7	0	0	0	0	0	0	0	0	0	0	0.4	1
S8	0	0	0	0	0	0	0	0	0	0.4	0.2	0.2
S9	0	0	0	0	0	0	0	0	0	0.4	0.4	0.4
S10	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	0

Общее влияние угроз на финансовые потери составит 1.2, на личную информацию – 1.8, на контроль над устройством – 2.4. Общий риск будет составлять 5.4. По полученным результатам можно сделать вывод, что общий риск в результате внедрения системы обнаружения вредоносных программ снизился на 4.2, что подтверждает целесообразность его применения в ОС для мобильных устройств.

Следовательно, риски от действия угроз на целевые факторы составят: R1--10=6; R1-11=20; R1-12=8; R2-10=0; R2-11=20; R2-12=20; R3-10=12; R3-11=20; R3-12=8; R4-10=6; R4-11=10; R4-12=4; R5-10=12; R5-11=20; R5-12=8.

Таким образом, общий риск с учетом значимости целевых факторов (v1= =0.39; v2=0.2; v3=0.41) составит: R=0.39(6+0+12+6+12)+0.2(20+20+12+6+12)+ +0.41(8+20+8+4+8)=47.7.

Следовательно, риск влияния угроз (с учетом общего риска) на целевые факторы составляет 47 %.

Эффективность внедренного средства защиты можно рассчитать по формуле

$$Эффективность = \frac{R_{до} - R_{после}}{R_{до}},$$

где $R_{до}$ – общий риск до внедрения системы;

$R_{после}$ – общий риск после внедрения системы.

Эффективность внедрения дополнительного средства защиты составляет 49 %, что позволило значительно повысить уровень защищенности ОС для мобильных устройств в целом.

4.5 Сравнение полученных в процессе эксперимента результатов с существующими антивирусными программами

С целью оценки эффективности работы разработанного исследовательского прототипа модели системы обнаружения вредоносных программ с точки зрения

практического применения необходимо выполнить сравнение результатов работы разработанной программы с результатами существующих антивирусных программ. Для выполнения сравнения были использованы сервисы securelist.ru, virusbtn.com и anti-malware.ru, на которых аккумулируется статистика работы антивирусных программ, в частности по обнаружению новых вредоносных программ. Измерялся процент распознавания новых вредоносных программ на 42 образцах антивирусных программ, результаты приведены в таблице 4.6. В качестве проверки были использованы 100 типов программ: модель исследовательского прототипа показала результат, который составил 93 %, что превысило лучший показатель антивирусной программы равный 80 %.

Таблица 4.6 – Сравнение результатов обнаружения

Название	Процент обнаружения
1	2
Система обнаружения вредоносных программ	93
AhnLab V3 Mobile Security 3.0	80
Alibaba Mobile Security 2.14	76
Antiy AVL 2.4	76
Avast Mobile Security 4.0	73
AVG AntiVirus Free 5.1	73
Avira Free Android Security 4.3	72
Baidu Mobile Security 5.10	71
Bitdefender Mobile Security 3.2	71
BullGuard Mobile Security 14.0	71
Cheetah Mobile Clean Master 5.11	65
Cheetah Mobile CM Security 2.8	64
ESET Mobile Security & Antivirus 3.2	64

Окончание таблицы 4.6

1	2
G Data Internet Security 25.9	63
Ikarus mobile.security 1.7	63
Kaspersky Lab Internet Security 11.9	61
Lookout Mobile Security & Antivirus 9.33	59
McAfee Mobile Security 4.5	58
Norton Mobile Security 3.13	58
NSHC Droid-X 3.0	55
G Data Internet Security 25.9	43

Весомым преимуществом антивирусных программ является поиск и обнаружение вредоносных программ при наличии тех в базе антивирусных сигнатур. Соответственно, количество ложных срабатываний снижается до минимума. Но в случае появления новых вредоносных программ ситуация изменяется в худшую сторону: процент обнаружения на порядок меньше, количество ложных срабатываний больше. Среди рассматриваемых антивирусных программ нет ни одной с уровнем распознавания выше 80 %. Средний процент обнаружения составил 65 %.

Таким образом, можно сделать вывод, что антивирусные программы обладают низкой эффективностью обнаружения новых вредоносных программ, результат лучшей антивирусной программы составил 80 %, средний результат по всем рассматриваемым образцам составил 65 %. Разработанный исследовательский прототип модели системы обнаружения вредоносных программ показал лучший результат в 93 % и сравнительно небольшое количество ложных срабатываний, равное 8 программам из 100 рассматриваемых образцов, что составило 17 %, но при этом не учитывался результат работы аппарата нечеткой логики, который выполнил уточнение и коррекцию результата путем дополнительной классификации. Машина опорных векторов ошиблась,

выдав вредоносную программу за безопасную, но нечеткая логика показала результат: подозрительная 45 % и опасная 65 %, таким образом дополнив работу машины опорных векторов и улучшив показатели эффективности обнаружения.

4.6 Перспективы дальнейшего применения разработанного метода обнаружения вредоносных программ

В настоящее время нейронные сети широко применяются в различных задачах, таких как классификация, распознавание, предсказание. В своем развитии они претерпели изменения и приобрели значительные преимущества, в частности в алгоритмах обучения. Разработанная нейронная сеть, названная ограниченной машиной Больцмана, обученная на основе алгоритма глубинного обучения, позволила увеличить процент распознавания образов на 13 % от лучшего имеющегося на сегодняшний день результата. Это объясняет скачок в развитии машинного обучения (искусственного интеллекта) в распознавании как образов, так и речи, например, голосовой сервис и сервис поиска по картинкам Google, виртуальный консультант Siri от Apple. Данные сервисы на высоком уровне справляются с поставленными перед ними задачами.

Разработанная методика обнаружения вредоносных программ в ОС для мобильных устройств (на примере Android) позволит увеличить эффективность обнаружения новых вредоносных программ. Данный алгоритм в связке с классическими методами анализа (сигнатурный метод) позволит значительно увеличить процент обнаружения как новых, так и уже известных вредоносных программ. Следовательно, это позволит укрепить существующие механизмы защиты информации в ОС для мобильных устройств, а также защитить личную информацию, которая располагается в мобильном устройстве.

4.7 Выводы по четвертой главе

1. Предложена архитектура исследовательского прототипа модели системы обнаружения вредоносных программ на основе проделанных исследовательских экспериментов, позволяющая выполнять обнаружение с учетом различных условий и помех, а также повысить эффективность обнаружения вредоносных программ, отличающаяся новым подходом.

2. Разработана программа исследовательского прототипа модели системы обнаружения вредоносных программ на языке C++ на основе двух методов: машины опорных векторов и нечеткой логики, позволяющая выполнять обнаружение вредоносных программ, вести анализ и статистику результатов, а также записывать все действия в журнале, отличающаяся высокой эффективностью обнаружения вредоносных программ.

3. На основе разработанной нечеткой когнитивной карты, позволяющей рассчитать риск и эффективность после внедрения системы обнаружения вредоносных программ, установлено, что риск нарушения информационной безопасности ОС для мобильных устройств снизился на 46 %, эффективность составила 49 %.

4. Сравнительный анализ результатов по обнаружению вредоносных программ показал, что лучший результат антивирусных программ составил 80 %, средний по всем рассматриваемым типам антивирусных программ – 65 %. Результат обнаружения разработанной программы составил 93 %. Ошибка первого рода составила 0 %, второго рода 17 %. Нечеткая логика выполнила дополнительную классификацию на ошибках второго рода (подозрительная 45 % и опасная 65 %), таким образом выполнив коррекцию результата классификации машины опорных векторов.

ЗАКЛЮЧЕНИЕ

В диссертационной работе поставлена и решена задача повышения эффективности обнаружения вредоносных программ с применением системы обнаружения вредоносных программ на основе машины опорных векторов и нечеткой логики.

В процессе выполнения работы были получены следующие результаты:

1. Показано, что проблема защищенности ОС для мобильных устройств является актуальной темой на основе анализа сложившейся ситуации, согласно которому число вредоносных программ выросло более чем в 10 раз за период с 2012 по 2014 гг. и составило 12 миллионов в 2014 году. Анализ архитектуры мобильной ОС, структуры прикладных программ и уязвимостей (вредоносные программы, система разрешений, модифицированная мобильная ОС и т.д.) позволил сделать вывод, что существует необходимость исследований в данном направлении.

2. Предложена и обоснована экспериментальная выборка, которая позволяет описать поведенческий характер потенциальной вредоносной программы, на основе исследования множества прикладных программ, позволяющая осуществить их классификацию.

3. Разработаны системные модели процесса функционирования системы обнаружения вредоносных программ для ОС Android с учетом различных факторов на основе SADT-методологии и IDEF-технологий, позволяющие интегрировать систему обнаружения вредоносных программ с компонентами внутренних и внешних механизмов защиты ОС Android.

4. Показано, что машина опорных векторов выполняет классификацию разработанной экспериментальной выборки с большей точностью (процент правильно классифицированных составил 72,3 %, количество ошибок I рода – 0 %, ошибок II рода – 27,7 %), чем классические и нейросетевые методы, на основе проделанных экспериментов, позволяющих осуществить выбор наилучшего метода классификации. Установлено, что классические методы

обладают невысокой точностью: ошибки I рода – 26,08 %, ошибки II рода – 22,7 %, а также, что ошибки классификации нейронными сетями присутствуют по всей выборке, но увеличиваются с ростом количества помех в экспериментальной выборке.

5. Разработан и реализован в виде программы комплекс алгоритмов обнаружения вредоносных программ в ОС Android на основе интеллектуальных технологий, позволяющий с высокой точностью выполнять обнаружение вредоносных программ, отличающийся динамическим анализом поведения вредоносной программы от классического сигнатурного метода.

6. Установлено, что исследовательский прототип модели системы обнаружения вредоносных программ с эффективностью 80 % выполняет обнаружение вредоносных программ на основе сравнительного анализа результатов обнаружения вредоносных программ: лучший по антивирусным программам составил 60 %, что говорит о том, что эффективность обнаружения прототипа выше на 20 %.

Предложенный метод с применением машины опорных векторов и нечеткой логики позволяет увеличить эффективность обнаружения вредоносных программ по их поведенческому характеру при условиях, когда трудно различить безопасную программу от вредоносной.

СПИСОК ЛИТЕРАТУРЫ

1. Аверкин, А. Н., Кузнецов, О. П., Кулинич, А. А., Титова, Н. В. Поддержка принятия решений в слабоструктурированных предметных областях. Анализ ситуаций и оценка альтернатив // Известия РАН. Теория и системы управления.– 2006. – № 3. – С. 139 – 149.
2. Автоматизированное проектирование информационно-управляющих систем. Системное моделирование предметной области: Учебное пособие / Г. Г. Куликов, А. Н. Набатов, А. В. Речкалов; Уфимск. гос. авиац. техн. ун-т. Уфа, 2003. – 104 с.
3. Академия Intel: Введение в разработку приложений для ОС Android [Электронный ресурс]. Режим доступа: http://www.intuit.ru/studies/courses/12643/1191/lecture/21980?page=2 (дата обращения: 18.06.2015).
4. Анализ угроз информационной безопасности современных мобильных систем / Жернаков С. В., Гаврилов Г. Н. // Интеграционные процессы науки XXI века: сборник статей Международной научно-практической конференции. Стерлитамак: РИЦ АМИ, 2015. С. 54 – 60.
5. Андрейчиков, А.В., Андрейчикова, О.Н. Интеллектуальные информационные системы: Учебник. – М.: Финансы и статистика, 2004. – 424 с.
6. Балашов, П. А., Кислов, Р. И., Безгузикова, В. П. Оценка рисков информационной безопасности на основе нечеткой логики // Защита информации. Конфидент. – № 5. – 2003. – С. 56 – 59.
7. Безобразов, С. В. Искусственные иммунные системы для защиты информации: обнаружение и классификация компьютерных вирусов / С. В. Безобразов, В. А. Головко // Научная сессия МИФИ «Нейроинформатика»: материалы Всеросс. науч. конф., МИФИ, Москва, 20 – 23 янв. 2008. – С. 23–27.
8. Безопасность мобильных устройств [Электронный ресурс]. Режим доступа: http://www.osp.ru/win2000/2014/01/13039202/ (дата обращения: 23.05.2015).

9. Борисов, В. В., Круглов, В. В., Федулов, А. С. Нечеткие модели и сети. – М.: Горячая линия – Телеком, 2007. – 284 с.
10. Борисов, В. В., Круглов, В. В., Федулов, А. С. Нечеткие модели и сети. – М.: Горячая линия – Телеком, 2007. – 230 с.
11. Борисов, В. В., Федулов, А. С. Обобщенные нечёткие когнитивные карты // Нейрокомпьютеры: разработка, применение, 2004. – №4.
12. Боровиков, В. П. Нейронные сети. STATISTICA Neural Networks: Методология и технологии современного анализа данных. – 2-е изд., перераб. и доп. – М.: Горячая линия – Телеком, 2008. – 392 с.
13. Боровикова, В. П. Нейронные сети. STATISTICA Neural Networks: Методология и технологии современного анализа данных. 2-е изд. – М.: Горячая линия – Телеком, 2008. – 272 с.
14. Бояркин, А., Набиев, Н. Анализ Simplelocker-a — вируса-вымогателя для Android [Электронный ресурс] М.: TM, 2014. Режим доступа: http://habrahabr.ru/company/pentestit/blog/237207/ (дата обращения: 23.08.2015).
15. Вапник, В. Н., Червоненкис, А. Я. Теория распознавания образов. – М.: Наука, 1974. – 176 с.
16. Васильев, В. И. [и др.]. Разработка модели обнаружения сигнатур атак на основе метода опорных векторов // Материалы XII Международной научно-практической конференции «Информационная безопасность-2012». Ч. 1. – Таганрог: Изд-во ТТИ ЮФУ, 2012. – С. 192-201.
17. Васильев, В. И. Алгоритмы принятия решения на основе нечеткой логики // Методические указания к лабораторной работе по курсу «Системы искусственного интеллекта» и «Интеллектуальные системы». Уфа, 2007. – С. 4 – 17.
18. Васильев, В. И. Интеллектуальные системы защиты информации: Учебное пособие. – М.: Машиностроение, 2013. – 82 с.
19. Васильев, В. И. Решение задачи распознавания образов с помощью нейронных сетей // Методические указания к лабораторной работе по курсу

«Системы искусственного интеллекта» и «Интеллектуальные системы». Уфа, 2007. – С. 4 – 15.

20. Васильев, В. И., Ильясов, Б. Г. Интеллектуальные системы управления: теория и практика: учебное пособие. – Уфа: УГАТУ, 2008. – 446 с.
21. Васильев, В. И., Кудрявцева, Р. Т. Анализ и управление информационной безопасностью вуза на основе когнитивного моделирования // Системы управления и информационные технологии, 2007, №1(27). – С. 74 – 81.
22. Валеев, С. С., Дьяконов, М. Ю., Нейросетевая система обнаружения аномального поведения вычислительных процессов микроядерных операционных систем: дис. канд. техн. наук: 05.13.19 – Уфа, 2010.
23. Вейтман В. Linux. Необходимый код и команды. Карманный справочник. – М.: Вильямс, 2015. – 237 с.
24. Вентцель, Е. С. Теория вероятностей. – М.: Наука, 1969. – 292 с.
25. Вентцель, Е. С. Исследование операций. – М.: Высшая школа, 2007. – 112 с.
26. Взгляд на защиту информации в мобильной системе / Жернаков С. В., Гаврилов Г. Н. // Научные перспективы XXI века: материалы XIII Международной (заочной) научно-практической. Нефтекамск: РИО ООО «Наука и образование», 2015. – С. 46–52.
27. Воронцов, К. В. Лекции по методу опорных векторов, 2007. – С. 2 –13.
28. Враг в телефоне [Электронный ресурс]. Режим доступа: https://securelist.ru/analysis/obzor/25150/vrag-v-telefone/ (дата обращения: 22.11.2014).
29. Выявление вредоносных программ с использованием современного интеллектуального метода на этапе установки связи / Жернаков С. В., Гаврилов Г. Н. // Актуальные вопросы развития инновационной деятельности в новом тысячелетии: XVII Международная научно-практическая конференция. Новосибирск, Ежемес. науч. журнал «Educatio», 2015. С. 9–13.
30. Вятченин, Д. А. Нечеткие методы автоматической классификации. Минск: УП Технопринт, 2004. – 134 с.

31. Галушкин, А. И. Нейрокомпьютеры в решении задач обеспечения информационной безопасности // Нейрокомпьютеры: разработка, применение – 2005. – №12. – С. 50 – 58.
32. Головко, В. А. Нейронные сети: обучение, организация и применение. Кн. 4: учеб. пособие для вузов. – М.: ИИРЖР, 2001. – 178 с.
33. Голощапов, А. Google Android. Системные компоненты и сетевые коммуникации. – СПб.: БХВ-Петербург, 2012. – 53 с.
34. ГОСТ 350922-96 «Защита информации. Основные термины и определения».
35. Гузаиров, М. Б., Машкина, И. В. Управление защитой информации на основе интеллектуальных технологий: учебное пособие. – М.: Машиностроение, 2013. – 241 с.
36. Детектирование вредоносного программного обеспечения в мобильной операционной системе на базе Android на основе разрешений с применением метода опорных векторов / Жернаков С. В., Гаврилов Г. Н. // Научно периодическое издание Ceteris Paribus: Европейский фонд инновационного развития. Ежемес. науч. журнал «Ceteris Paribus», 2015.С. 10 –14.
37. Донцова, Л., Донцов, Е. Сравнение метода опорных векторов и нейронной сети при прогнозировании банкротства предприятий. [Электронный ресурс]. Режим доступа: http://urf.podelise.ru/docs/1100/index-78995.html (дата обращения: 08.09.2015).
38. Дюк, В. А., Самойленко, А. П. Data mining: учебный курс – СПб.: Питер, 2001. – 412 с.
39. Жуковин, В. Е. Нечеткие многокритериальные модели принятия решений. – М.: Наука, 1992. – 6 с.
40. Зайченко, Ю. П. Нечеткие модели и методы в интеллектуальных системах. Учебное пособие для студентов высших учебных заведений. – К.: Слово, 2008. – 91 с.
41. Зак, Ю. А. Принятие решений в условиях нечетких и размытых данных: Fuzzy-технологии. – М.: ЛИБРОКОМ, 2013 – 179 с.

42. Кеба, И. В. Диагностика авиационных газотурбинных двигателей. – М.: Транспорт, 1980. – 184 с.
43. Кельберт, М. Я., Сухов, Ю. М. Вероятность и статистика в примерах и задачах. Т. II: Марковские цепи как отправная точка теории случайных процессов и их приложения. – М.: МЦНМО, 2009. – 209 с.
44. Кластерный анализ (кластеризация) [Электронный ресурс]. Режим доступа: http://statistica.ru/glossary/general/klasternyy-analiz-klasterizatsiya/ (дата обращения: 01.09.2015).
45. Колисниченко, Д. Программирование для Android. Самоучитель. – СПб.: БХВ-Петербург, 2012. – 32 с.
46. Корелов, С. В. Обнаружение аномального поведения процесса на основе системы вызовов/ С. В. Корелов, А. С. Пуров, Л. Ю. Ротков: материалы конф./ Научная конф. По радиофизике. НГУ, 2006 [Электронный ресурс]. – Режим доступа: http://www.rf.unn.ru/rus/sci/books/06/doc/11InfSys06.doc.
47. Корт, С. С. Теоретические основы зашиты информации: Учебное пособие. – М.: Гелиос АРВ, 2004. – 240 с.
48. Котельников, Е., Козвонина, А. Параллельная реализация машины опорных векторов с использованием методов кластеризации [Электронный ресурс]. Режим доступа: http://ict.informika.ru/vconf/files/11508.pdf (дата обращения: 03.09.2015).
49. Кудрявцева, Р. Т. Управление информационными рисками с использованием технологий когнитивного моделирования (на примере высшего учебного заведения): дис. канд. техн. наук: 05.13.19 – Уфа, 2008. – 142 с.
50. Леоненков, А. В. Нечеткое моделирование в среде MATLAB и fuzzyTECH. – СПб.: БХВ-Петербург, 2005 – 399 с.
51. Любимов, Н., Михеев, Е., Лукин, А. Сравнение алгоритмов кластеризации в задаче диктора [Электронный ресурс]. Режим доступа: http://www.researchgate.net/publication/267690636 (дата обращения: 03.09.2015).

52. Маклаков, С.В. Моделирование бизнес-процессов с BPwin 4.0. – М.: ДИАЛОГМИФИ, 2002 – 22 с.
53. Матвеев, Е. Программирование под Android. – СПб.: Питер. 2010 – 71 с.
54. Машкина, И. В., Васильев, В.И., Миронов, К. В., Шарабыров, И. В. Разработка модели обнаружения сигнатур атак на основе метода опорных векторов // Материалы XII Международной научно-практической конференции «Информационная безопасность-2012». Ч. 1. – Таганрог: Изд-во ТТИ ЮФУ, 2012. – С. 192-201.
55. Машнин, Т. Современные Java-технологии на практике. – СПб.: БХВ-Петербург, 2006. – 82 с.
56. Миронов, К. В., Шарабыров, И. В. О применении метода опорных векторов в системах обнаружения атак // Мавлютовские чтения: Всероссийская молодежная научная конференция: сборник трудов в 5 т. Т. 3. – УГАТУ, 2012. – С. 28–30.
57. Мобильные угрозы [Электронный ресурс]. Режим доступа: http://www.kaspersky.ru/internet-security-center/threats/mobile (дата обращения: 01.10.2014).
58. Нейронные сети [Электронный ресурс]. Режим доступа: http://www.statlab.kubsu.ru/sites/project_bank/nural.pdf (дата обращения: 14.11.2015).
59. Нейронные сети [Электронный ресурс]. Режим доступа: http://www.statsoft.ru/home/textbook/modules/stneunet.html (дата обращения: 15.11.2015).
60. Об одном подходе защиты информации в средствах мобильной связи / Жернаков С. В., Гаврилов Г. Н. // XV Международная научно-практическая конференция: Научное обозрение физико-математических и технических наук в XXI веке. Москва: Ежемес. науч. журнал «Prospero», 2015.С. 9-13.
61. Обзор фрагментов кода самых популярных типов вирусов [Электронный ресурс]. Режим доступа: https://xakep.ru/2014/09/18/android-malware-source/ (дата обращения: 22.09.2014).

62. Обнаружение вредоносных интернет-страниц на основе технологии нейронных сетей / Котов В. Д. // Вестник УГАТУ: науч. журнал Уфимск. гос. авиац. техн. ун-та. 2012. –Т. 16, № 8 (53). – С. 73–79.
63. Острейковский, В. А. Эксплуатация атомных станций: Учебник для вузов. – М.: Энергоатомиздат, 1999. – 600 с.
64. Питерсон, Дж. Теория сетей Петри и моделирование систем: Пер. с англ. – М.: Мир, 1984. – 33 с.
65. Портенко, Н. И., Скороход, А. В., Шуренков, В. М. Марковские процессы. – ВИНИТИ, 1989. – 178 с.
66. Пострадали десятки тысяч клиентов «Сбербанка» по всей стране [Электронный ресурс]. Режим доступа: https://hi-tech.mail.ru/news/sbarbank-android-users-hacked-and-robbed-by-trojan.html (дата обращения: 01.12.2014).
67. Применение искусственной иммунной системы для обнаружения вредоносных программ в мобильных устройствах Android / Жернаков С. В., Гаврилов Г. Н. // XXII Туполевские чтения (школа молодых ученных): Международная молодежная научная конференция. Том IV. Казань: Изд-во «Фолиант», 2015. С. 70-78.
68. Реализация метода опорных векторов в системе Android для классификации и обнаружения вредоносных программ / Жернаков С. В., Гаврилов Г. Н. // Научные перспективы XXI века. Достижения и перспективы нового столетия: XVII Международная научно-практическая конференция. Новосибирск: Ежемес. науч. журнал «МИС» №6 (17), 2015. С. 10–14.
69. Рик Роджерс, Джон Ломбардо, Зигурд Медниекс, Блейк Мейк. Android. Разработка приложений. – М.: ЭКОМ Паблишерз, 2010. – 85 с.
70. Ротштейн, А. П. Интеллектуальные технологии идентификации: нечеткая логика, генетические алгоритмы, нейронные сети. Винница: УНИВЕРСУМ—Винница, 1999. – 320 с.
71. Рутковская, Д., Пилиньский, М., Рутковский, Л. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. – М.: Горячая линия – Телеком, 2006. – 366 с.

72. Сатия Коматинени, Дэйв Маклин, Саид Хашими. Android 3 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов. – М.: Вильямс, 2012. – 36 с.
73. Свечников, Л. А. Интеллектуальная система обнаружения атак на основе имитационного моделирования с применением нечетких когнитивных карт: дис. канд. техн. наук: 05.13.19 – Уфа, 2010. – 127 с.
74. Силов, В. Б. Принятие стратегических решений в нечеткой обстановке. – М.: ИНПРО – РЕС, 1995. – 118 с.
75. Сравнения антивирусов, DLP и других средств защиты [Электронный ресурс]. Режим доступа: http://www.anti-malware.ru/compare ?????!!!! (дата обращения: 18.06.2015).
76. Старовойтов А. Настройка аппаратных средств в Linux. – СПб.: БХВ-Петербург, 2006. – 100 с.
77. Стратонович, Р. Л. Условные марковские процессы и их применение к теории оптимального управления. – М.: МГУ, 1966. – 25 с.
78. Таганов, К. В., Овчаров, Л. А., Тырышкин, А. Н. Аналитические методы исследования систем. – М.: Советское радио, 1974. – 32 с.
79. Трахтенгерц, Э. А. Компьютерная поддержка принятия решений: научно-практич. издание. Серия: Информатизация Россия на пороге XXI века. – М.: СИНТЕГ, 1998. – 376 с.
80. Усков, А. А., Кузьмин, А. В. Интеллектуальные технологии управления. Искусственные нейронные сети и нечеткая логика. – М.: Горячая линия – телеком, 2004. – 49 с.
81. Уязвимости платформы Android. Настоящее и будущее [Электронный ресурс]. Режим доступа: http://habrahabr.ru/company/drweb/blog/142993/ (дата обращения: 28.11.2014).
82. Федотова, Д. Э., Семенов, Ю. Д., Чижик, К. Н. CASE-технологии: Практикум. – М.: Горячая линия – Телеком, 2005. – 63 с.
83. Филин, С. А. Информационная безопасность: Учебное пособие. – М.: Альфа-Пресс, 2006. – 412 с.

84. Хайкин, С. Нейронные сети: полный курс, 2 – е издание.: Пер. с англ. – М: Издательский дом “Вильямс”, 2006. – 414 с.
85. Халафян, А. А. STATISTICA 6. Статистический анализ данных. 3-е изд. Учебник. – М.: Бином-Пресс, 2007. – 241 с.
86. Хоффман, Л. Дж. Современные методы защиты информации. / Под ред. В.А. Герасименко. – М.: Советское радио, 1980. – 257 с.
87. Цуканова, О. А. Методология и инструментарий моделирования бизнес-процессов: учебное пособие – СПб.: Университет ИТМО, 2015. – 12 с.
88. Черезов, Д., Тюкачев, Н. Обзор основных методов классификации и кластеризации данных [Электронный ресурс] Воронеж: Вестник ВГУ, 2009. ТМ, 2014.. Режим доступа: http://www.vestnik.vsu.ru/pdf/analiz/2009/02/2009-02-05.pdf (дата обращения: 05.09.2015).
89. Черемных, С. В., Семенов, И. О., Ручкин, В. С. Моделирование и анализ систем. IDEF-технологии: практикум. – М.: Финансы и статистика, 2006. – 25 с.
90. Штовба, С. Д. Введение в теорию нечетких множеств и нечеткую логику. Винница: Континент – ПРИМ, 1996. – 132с.
91. Шумский, А. Ю. Зашита компьютерной информации от несанкционированного доступа. – СПб: Наука и техника, 2005. – 224 с.
92. Брюхомицкий, Ю. А. Макаревич, О. Б. Обзор исследований и разработок по информационной безопасности // XII Международная научно-практическая конференция «Информационная безопасность-2012». 25–29 июня 2012 г., Таганрог, Россия. – С. 8 – 21.
93. Юрий Лифшиц. Метод опорных векторов. Лекция №7 курса Алгоритмы для Интернета, 2006. – С. 1-9.
94. Ярочкин, В. И. Информационная безопасность: Учебник для студентов вузов. – М.: Академический Проспект; Гаудеамус, 2-е изд. 2004. – 544 с.
95. Abhishek Dubey, Anmol Misra. Android Security. Attacks and defenses. NY, CRCPress, 2013. 17 p.

96. Android 4.3 и SELinux [Электронный ресурс]. Режим доступа: https://securelist.ru/blog/issledovaniya/3394/android-4-3-i-selinux/ (дата обращения: 18.06.2015).
97. Android. (n.d.). *Introduction to Android*. Retrieved 1 23, 2014, from Android Developers [Электронный ресурс]. Режим доступа: http://developer.android.com/guide/index.html (дата обращения: 23.01.2015).
98. Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS Symposium 2014, Switzerland, 2014, vol. 4, no. 1 Available at: https://user.informatik.unigoettingen.de/~krieck/docs/2014-ndss.pdf (Accessed 08 March 2015).
99. AV-Comparatives: Антивирусы для Android: Март 2015 [Электронный ресурс]. Режим доступа: http://www.comss.ru/page.php?id=2424 (дата обращения: 29.09.2015).
100. CVE-2009-1185 [Электронный ресурс]. Режим доступа: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1185 (дата обращения: 22.11.2015).
101. CVE-2011-1823 [Электронный ресурс]. Режим доступа: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1823 (дата обращения: 22.11.2015).
102. Fan Yuhui, Xu Ning. The Analysis of Android Malware Behaviors. International Journal of Security and Its Applications, Australia, 2015, vol. 9, no. 3 Available at: http://www.sersc.org/journals/IJSIA/vol9_no3_2015/25.pdf (Accessed 08 March 2015).
103. Global Smartphone Adoption [Электронный ресурс]. Режим доступа: http://en.wikipedia.org/wiki/ Mobile_operating_system (дата обращения: 22.09.2014).
104. Joseph Annuzzi, Jr. Lauren Darcey, Shane Conder. Introduction to Android application development. Android essentials. Fourth edition. Michigan, Addison-Wesley. 2013. 97 p.

105. Karim Yaghmour. Embedded Android. US, O'REILLY. 2013. 21 p.
106. Kaspersky Security Bulletin 2014. Основная статистика за 2014 год [Электронный ресурс]. Режим доступа: http://securelist.ru/analysis/24580/kaspersky-security-bulletin-2014-osnovnaya-statistika-za-2014-god/ (дата обращения: 01.12.2014).
107. Kaspersky Security Bulletin 2014. Прогнозы на 2015 год https://securelist.ru/analysis/24575/kaspersky-security-bulletin-2014-prognozy-na-2015-god/
108. Ken Dunham, Shane Hartman, Manu Quintans, Jose Andre Morales, Tim Strazzere. Android Malware and Analysis. NY, CRCPress, 2015. 91 p.
109. Malware bump [Электронный ресурс]. Режим доступа: http://contagiodump.blogspot.com.tr (дата обращения: 26.07.2014).
110. Nikolay Elenkov. Android security internals. An In-Depth Guide to android's security architecture. San Francisco, No Starch Press, Inc, 2015. 21 p.
111. Sanz B., Santos I., Nieves J., Laorden C., Alonso-Gonzalez I., G. Bringas P. MADS: Malicious android applications detection through string analysis. Network and System Security, Springer Berlin Heidelberg, 2011, vol. 5, no. Available at: http://www.researchgate.net/publication/256194745_MADS_Malicious_Android_Applications_Detection_through_String_Analysis (Accessed 08 March 2015).
112. Security Android [Электронный ресурс]. Режим доступа: https://source.android.com/devices/tech/security/index.html (дата обращения: 18.06.2015).
113. Sheran Gunasekera. Android Apps Security. Apress, 2012. 3 p.
114. Six J. Application Security for the Android Platform. Processes, Permissions, and Other Safeguards. CA, O'Reilly Media, 2011. 2 p.
115. W. Frank Ableson, Robi Sen, Chris King. Android in action. Second Edition. US, Manning Publications Co. 2011. 34 p.
116. Smartphone OS Market Share, 2016 Q3 [Электронный ресурс]. Режим доступа: https://www.idc.com/promo/smartphone-market-share/os (дата обращения: 19.07.2016).

117. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 1st quarter 2017 [Электронный ресурс]. Режим доступа: https://www.idc.com/promo/smartphone-market-share/os (дата обращения: 25.01.2017).

118. Market Share Statistics for Internet Technologies [Электронный ресурс]. Режим доступа: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1 (дата обращения: 25.01.2017).

ПРИЛОЖЕНИЕ А

Список разрешений на работу с сервисами мобильной системы

№	Название разрешения	Описание
1	ACCESS_CHECKIN_PROPERTIES	Предоставляет доступ на чтение или запись параметров в базы данных, чтобы обновлять загруженные данные.
2	ACCESS_COARSE_LOCATION	Позволяет программе получать доступ к приблизительному местоположению посредством Cell-ID или Wi-Fi.
3	ACCESS_FINE_LOCATION	Позволяет программе получать точное местоположение посредством GPS.
4	ACCESS_LOCATION_EXTRA_COMMANDS	Позволяет программе получать дополнительные функции, связанные с местоположением.
5	ACCESS_MOCK_LOCATION	Позволяет программе создавать ненастоящее местоположение для своих целей или проверки.
6	ACCESS_SURFACE_FLINGER	Позволяет программе использовать функции композитного менеджера графических слоёв.
7	ACCESS_NETWORK_STATE	Позволяет программе получать информацию о состоянии сетей.
8	ACCESS_WIFI_STATE	Позволяет программе получать информацию о состоянии Wi-Fi сети.
9	ACCOUNT_MANAGER	Дает программе возможность управлять логинами (пользователи).
10	ADD_VOICEMAIL	Позволяет программе добавлять голосовые сообщения.
11	AUTHENTICATE_ACCOUNTS	Дает возможность программе выступать в качестве аутентификатора учетных записей для менеджера учетных записей.
12	BATTERY_STATS	Предоставляет возможность программе собирать статистику по батарее.
13	BIND_ACCESSIBILITY_SERVICE	Позволяет программе сообщить сервису специальных возможностей, какая программа может получить доступ к данным к

		специальным возможностям.
14	BIND_APPWIDGET	Позволяет программе сообщить сервису виджетов, какая программа может получить доступ к данным в виджетов.
15	BIND_CARRIER_MESSAGING_SERVICE	Запрос службы, которая принимает уведомления из системы о новых смс и ммс сообщениях.
16	BIND_DEVICE_ADMIN	Должен требоваться приемником администратора устройства, гарантировать, что только система может взаимодействовать с ним.
17	BIND_DREAM_SERVICE	Позволяет запрашивать сервис управления сном.
18	BIND_INPUT_METHOD	Позволяет запрашивать сервис методов ввода.
19	BIND_NFC_SERVICE	Позволяет запрашивать NFC сервис.
20	BIND_NOTIFICATION_LISTENER_SERVICE	Позволяет запрашивать службу уведомления.
21	BIND_PRINT_SERVICE	Позволяет запрашивать сервис печати.
22	BIND_REMOTEVIEWS	Позволяет запрашивать сервис удаленного просмотра.
23	BIND_TEXT_SERVICE	Позволяет запрашивать текстовый сервис.
24	BIND_TV_INPUT	Позволяет запрашивать сервис входа TV.
25	BIND_VOICE_INTERACTION	Позволяет запрашивать голосовую службу взаимодействия.
26	BIND_VPN_SERVICE	Позволяет запрашивать vpn сервис.
27	BIND_WALLPAPER	Позволяет запрашивать сервис обоев.
28	BLUETOOTH	Позволяет соединяться с другими устройствами через bluetooth.
29	BLUETOOTH_ADMIN	Позволяет самостоятельно находить и сопрягаться с bluetooth-устройствами.
30	BLUETOOTH_PRIVILEGED	Управление в bluetooth в фоновом режиме.
31	BODY_SENSORS	Доступ к сенсорам измерения состояния тела.
32	BRICK	Не используется.
33	BROADCAST_PACKAGE_REMOVED	Позволяет вещать уведомление о том, что пакет программы был удален.

34	BROADCAST_SMS	Позволяет передавать отчет о доставке смс-сообщения.
35	BROADCAST_STICKY	Позволяет передавать уведомление о заметках.
36	BROADCAST_WAP_PUSH	Позволяет передавать отчет о доставке при отправке данных посредством WAP.
37	CALL_PHONE	Право инициировать телефонный звонок, минуя стандартный пользовательский интерфейс набора номера.
38	CALL_PRIVILEGED	Позволяет вызывать любой телефонный номер, в том числе номера экстренных служб, минуя стандартный пользовательский интерфейс набора номера.
39	CAMERA	Доступ к камере устройства.
40	CAPTURE_AUDIO_OUTPUT	Право захватывать аудио выход.
41	CAPTURE_SECURE_VIDEO_OUTPUT	Право захватывать защищенный видео выход.
42	CAPTURE_VIDEO_OUTPUT	Право захватить видео выход.
43	CHANGE_COMPONENT_ENABLED_STATE	Позволяет программе активировать или деактивировать компоненты других программ.
44	CHANGE_CONFIGURATION	Право изменять текущую конфигурацию, например, местоположение.
45	CHANGE_NETWORK_STATE	Позволяет менять состояние подключения к сети.
46	CHANGE_WIFI_MULTICAST_STATE	Право входить в режим Wi-Fi мультивещание, многоадресное вещание.
47	CHANGE_WIFI_STATE	Позволяет изменить состояние Wi-Fi подключения.
48	CLEAR_APP_CACHE	Право очищать кэш всех установленных на устройстве программ.
49	CLEAR_APP_USER_DATA	Право очистить пользовательские данные, связанные с программами.
50	CONTROL_LOCATION_UPDATES	Право включать или выключать уведомление об обновлении местоположения.
51	DELETE_CACHE_FILES	Право удалять файлы кэша.

52	DELETE_PACKAGES	Позволяет удалять пакеты.
53	DEVICE_POWER	Доступ к управлению питанием устройства.
54	DIAGNOSTIC	Проводить диагностику системных ресурсов.
55	DISABLE_KEYGUARD	Право отключать блокировку клавиатуры.
56	DUMP	Право получать информации о системных службах.
57	EXPAND_STATUS_BAR	Право сворачивать или разворачивать строку состояния.
58	FACTORY_TEST	Запуск тестовой программы производителя, работает в качестве привилегированного пользователя.
59	FLASHLIGHT	Доступ к вспышке камеры.
60	FORCE_BACK	Позволяет использовать команду «Назад».
61	GET_ACCOUNTS	Позволяет получить доступ к списку учетных записей в сервисе учетных записей.
62	GET_PACKAGE_SIZE	Право узнавать занимаемое любым пакетом место.
63	GET_TASKS	Право получать информацию о запущенных сейчас или недавно программах и сервисах.
64	GET_TOP_ACTIVITY_INFO	Позволяет получать информацию о текущих или прошлых задачах, процессах.
65	GLOBAL_SEARCH	Права чтобы глобальная поисковая система получила доступ к данным пользователя.
66	HARDWARE_TEST	Доступ к аппаратной периферии.
67	INJECT_EVENTS	Позволяет вставлять события по вводу (нажатие клавиш, касания, движение трекбола) в общий поток событий для любого окна.
68	INSTALL_LOCATION_PROVIDER	Позволяет устанавливать местонахождение поставщика в расположение сети оператора.
69	INSTALL_PACKAGES	Право устанавливать пакеты.
70	INSTALL_SHORTCUT	Позволяет установить ярлык.
71	INTERNAL_SYSTEM_WINDOW	Право открывать окна, которые используются системным пользовательским интерфейсом.
72	INTERNET	Позволяет открывать сетевые сокеты.
73	KILL_BACKGROUND_PROCESSES	Право вызывать процесс служащий для очистки памяти.

74	LOCATION_HARDWARE	Позволяет программе использовать функции аппаратного местоположения (управление геозонами).
75	MANAGE_ACCOUNTS	Позволяет управлять списком учетных записей в менеджере учетных записей.
76	MANAGE_APP_TOKENS	Позволяет создавать, удалять, сортировать список программ в менеджере окон.
77	MANAGE_DOCUMENTS	Доступ для управления доступом к документам.
78	MASTER_CLEAR	Не для использования сторонних программ.
79	MEDIA_CONTENT_CONTROL	Право отслеживать воспроизведение медиа файлов и управлять ими.
80	MODIFY_AUDIO_SETTINGS	Позволяет настраивать глобальные аудио настройки.
81	MODIFY_PHONE_STATE	Изменять состояние телефонной части: питание и т.д.
82	MOUNT_FORMAT_FILESYSTEMS	Права форматировать файловые системы съемных накопителей.
83	MOUNT_UNMOUNT_FILESYSTEMS	Права монтировать и демонтировать файловые системы съемных накопителей.
84	NFC	Даёт право пользователя выполнять операции ввода-вывода посредством NFC.
85	PERSISTENT_ACTIVITY	Не используется.
86	PROCESS_OUTGOING_CALLS	Право анализировать, изменять или отменять исходящие вызовы.
87	READ_CALENDAR	Доступ к чтению календарных записей пользователя.
88	READ_CALL_LOG	Позволяет просматривать журнал звонков пользователя.
89	READ_CONTACTS	Позволяет просматривать данные о контактах пользователя.
90	READ_EXTERNAL_STORAGE	Позволяет считывать с устройства внешнего хранения.
91	READ_FRAME_BUFFER	Доступ к кадровому буферу, в том числе позволяет программе делать скриншоты.

92	READ_HISTORY_BOOKMARKS	Право считывать историю и закладки браузера.
93	READ_INPUT_STATE	Не используется.
94	READ_LOGS	Право читать системные файлы журнала.
95	READ_PHONE_STATE	Позволяет программе получать информацию о состоянии телефона.
96	READ_PROFILE	Право считывать данные пользовательского профиля.
97	READ_SMS	Право читать смс-сообщения.
98	READ_SOCIAL_STREAM	Позволяет считывать информацию с пользовательского социального стрима.
99	READ_SYNC_SETTINGS	Позволяет просматривать настройки синхронизации.
100	READ_SYNC_STATS	Право считывать статистику синхронизации.
101	READ_USER_DICTIONARY	Право считывать словарь пользователя.
102	READ_VOICEMAIL	Право на чтение голосовых сообщений.
103	REBOOT	Позволяет перезагружать устройство.
104	RECEIVE_BOOT_COMPLETED	Право получать строку, которая вещается по окончанию загрузки системы.
105	RECEIVE_MMS	Право получать информацию о входящих ммс сообщениях, записывать и обрабатывать их.
106	RECEIVE_SMS	Право получать информацию о входящих смс-сообщениях, записывать и обрабатывать их.
107	RECEIVE_WAP_PUSH	Право анализировать входящие WAP-сообщения.
108	RECORD_AUDIO	Право записывать звук.
109	REORDER_TASKS	Право изменять порядок задач.
110	RESTART_PACKAGES	Не используется
111	SEND_RESPOND_VIA_MESSAGE	Позволяет программе телефон выполнять запросы к другим программам во время входящих звонков.
112	SEND_SMS	Права для отправки смс сообщений.
113	SET_ACTIVITY_WATCHER	Позволяет анализировать и контролировать, как в системе глобально запускаются различные задачи.

114	SET_ALARM	Позволяет создавать будильник или другое событие.
115	SET_ALWAYS_FINISH	Позволяет программе контролировать, будет ли вся активность немедленно приостановлена в фоновом режиме.
116	SET_ANIMATION_SCALE	Право изменять глобальный коэффициент масштабирования анимации.
117	SET_DEBUG_APP	Дает право настраивать программу для отладки.
118	SET_ORIENTATION	Даёт доступ к управлению ориентацией и поворотами экрана.
119	SET_POINTER_SPEED	Доступ к управлению скоростью курсора.
120	SET_PREFERRED_APPLICATIONS	Не используется
121	SET_PROCESS_LIMIT	Право устанавливать максимальное количество процессов, которые могут быть запущены.
122	SET_TIME	Позволяет устанавливать время.
123	SET_TIME_ZONE	Позволяет устанавливать часовой пояс.
124	SET_WALLPAPER	Позволяет устанавливать фон.
125	SET_WALLPAPER_HINTS	Позволяет устанавливать обо на рабочем столе.
126	SIGNAL_PERSISTENT_PROCESSES	Право запрашивать передачу полученного сигнала всем постоянным процессам.
127	STATUS_BAR	Право закрывать, открывать или полностью отключать панель статуса и иконок.
128	SUBSCRIBED_FEEDS_READ	Право считывать класс служащий для обмена данных между программами.
129	SUBSCRIBED_FEEDS_WRITE	Право изменять класс служащий для обмена данных между программами.
130	SYSTEM_ALERT_WINDOW	Позволяет открывать окна поверх всех других программ.
131	TRANSMIT_IR	Позволяет использовать ИК-передатчик устройства, если он доступен.
132	UNINSTALL_SHORTCUT	Право удалять ярлыки на рабочем столе.
133	UPDATE_DEVICE_STATS	Право обновлять статистику устройства.
134	USE_FINGERPRINT	Позволяет использовать оборудование сенсора.

135	USE_SIP	Дает право использовать программе сервис SIP — протокол установления сеанса.
136	VIBRATE	Дает доступ к управлению вибрацией.
137	WAKE_LOCK	Позволяет программе использовать режим сна, чтобы не выключать экран.
138	WRITE_APN_SETTINGS	Дает права записывать, но не считывать настройки точки доступа к сети Интернет.
139	WRITE_CALENDAR	Дает права записывать, но не считывать данные в календарь пользователя.
140	WRITE_CALL_LOG	Дает права записывать, но не считывать журнал звонков пользователя.
141	WRITE_CONTACTS	Позволяет записывать, но не считывать информацию о контактах пользователя.
142	WRITE_EXTERNAL_STORAGE	Позволяет использовать внешнюю память.
143	WRITE_GSERVICES	Права записывать данные в карты Google.
144	WRITE_HISTORY_BOOKMARKS	Дает права программе записывать, но не считывать данные в историю и закладки браузера.
145	WRITE_PROFILE	Права записывать, но не считывать данные в персональный профиль пользователя.
146	WRITE_SECURE_SETTINGS	Позволяет программе считывать и записывать данные в настройках безопасности системы.
147	WRITE_SETTINGS	Позволяет просматривать или изменять настройки системы.
148	WRITE_SMS	Позволяет программе писать смс сообщения.
149	WRITE_SOCIAL_STREAM	Позволяет записывать (но не читать) потоковые социальные данные пользователя.
150	WRITE_SYNC_SETTINGS	Позволяет программе настраивать синхронизацию.
151	WRITE_USER_DICTIONARY	Позволяет программе записывать данные в словарь пользователя.
152	WRITE_VOICEMAIL	Позволяет читать голосовые сообщения.

ПРИЛОЖЕНИЕ Б

Листинг алгоритмов машины опорных векторов и нечеткой логики программы

1. Листинг нечеткой логики (алгоритм Мамдани)

Файл: conclusion.h

```
#pragma once
#include <QString>
namespace Mamdani
{
 // Заключение правила. Например: Condition1 And Condition2 Then Conclusion1
 struct Conclusion
 {
  Conclusion(QString variableName, QString fuzzySetName)
   : VariableName(variableName)
   , FuzzySetName(fuzzySetName)
  {
  }
  QString VariableName; // Имя лингвистической переменной
  QString FuzzySetName; // Имя одного из термов лингвистической переменной
 };
}
```

condition.h

```
#pragma once
#include <QString>
namespace Mamdani
{
 // Условие правила. Например: Condition1 And Condition2 Then Conclusion1
 struct Condition
 {
  Condition(QString variableName, QString fuzzySetName)
   : VariableName(variableName)
   , FuzzySetName(fuzzySetName)
  {
  }
  QString VariableName; // Имя лингвистической переменной
  QString FuzzySetName; // Имя одного из термов лингвистической переменной
```

```
  };
}
```

Файл: fuzzyset.h

```
#pragma once
#include <QHash>
#include <QString>
#include <QSharedPointer>
#include <QSet>
namespace Mamdani
{
  /*
   * Терм лингвистической переменной
   */
  // Нечеткая переменная — это кортеж вида <α, X, A>, где:
  // α — имя нечеткой переменной;
  // X — её область определения;
  // A — нечеткое множество на универсуме X.

  class FuzzySet // is Term of Variable
  {
  public:
    enum Kind //@refactoring: make subclasses
    {
      Z_Function, //Z
      P_Function, //П
      L_Function, //L
      S_Function, //S
      Custom,
    };
    enum Point
    {
      A, B, C, D,
      END_POINT
    };
    typedef QPair<double, double> Range;
  public:
    explicit FuzzySet(Kind kind, QString name, QHash<Point, double> points, Range
range);
    Kind getKind() const;
```

```
    QString getName() const;
    QHash<Point, double> getPoints() const;
    Range getRange();
  public:
    // возвращает y [0;1] - координату для входного значения
    double getValue(double value);
    // вовзращает x координаты, в которых границы множетва пересекаются с графиком f = (y);
    QSet<double> getIntersections(double y); // return x coord  where fuzzy bound intersected with y.
  private:
    Kind TheKind;                // тип функции принадлежности Z или П или L или S
    QString TheName;             // α — имя нечеткой переменной;
    QHash<Point, double> ThePoints; // A — нечеткое множество на универсуме X. Ключевые точки по которым строится нечеткое множество.
    Range TheRange;              // X — её область определения;
  };
  typedef QSharedPointer<FuzzySet> FuzzySetPtr;
  typedef QList<FuzzySetPtr> FuzzySetList;
}
```

Файл: fuzzyset.cpp

```
#include "logic/mamdani/fuzzyset.h"
namespace
{
  double z_Function(double x, double b, double c)
  {
    if (x <= b)
      return 1;
    if (b < x && x <= c)
      return (c - x) / (c - b);
    return 0;
  }
  double p_Function(double x, double a, double b, double c, double d)
  {
    if (x <= a)
      return 0;
    if (a < x && x < b)
      return (x - a) / (b - a);
    if (b <= x && x <= c)
```

```
    return 1;
   if (c < x && x < d)
    return (d - x) / (d - c);
   return 0;
  }
  double l_Function(double x, double a, double b, double c)
  {
   if (x <= a)
    return 0;
   if (a < x && x <= b)
    return (x - a) / (b - a);
   if (b < x && x < c)
    return (c - x) / (c - b);
   return 0;
  }
  double s_Function(double x, double a, double b)
  {
   if (x <= a)
    return 0;
   if (a < x && x <= b)
    return (x - a) / (b - a);
   return 1;
  }
}
Mamdani::FuzzySet::FuzzySet(Mamdani::FuzzySet::Kind kind, QString name,
QHash<Point, double> points, Range range)
  : TheKind(kind)
  , TheName(name)
  , ThePoints(points)
  , TheRange(range)
{
}
Mamdani::FuzzySet::Kind Mamdani::FuzzySet::getKind() const
{
  return TheKind;
}
QString Mamdani::FuzzySet::getName() const
{
  return TheName;
```

```
}
QHash<Mamdani::FuzzySet::Point, double> Mamdani::FuzzySet::getPoints() const
{
 return ThePoints;
}
Mamdani::FuzzySet::Range Mamdani::FuzzySet::getRange()
{
 return TheRange;
}
double Mamdani::FuzzySet::getValue(double value)
{
 switch (TheKind)
 {
 case Z_Function:
  return z_Function(value, ThePoints[B], ThePoints[C]);
 case P_Function:
  return p_Function(value, ThePoints[A], ThePoints[B], ThePoints[C], ThePoints[D]);
 case L_Function:
  return l_Function(value, ThePoints[A], ThePoints[B], ThePoints[C]);
 case S_Function:
  return s_Function(value, ThePoints[A], ThePoints[B]);
 };
 return 0;
}
QSet<double> Mamdani::FuzzySet::getIntersections(double y)
{
 QSet<double> result;
 switch (TheKind)
 {
 case Z_Function:
    {
     result.insert(ThePoints[A]);
     result.insert(ThePoints[C] - (ThePoints[C] - ThePoints[B]) * y);
    }
    break;
 case P_Function:
    {
     result.insert(ThePoints[A] + (ThePoints[B] - ThePoints[A]) * y);
     result.insert(ThePoints[D] - (ThePoints[D] - ThePoints[C]) * y);
```

```
    }
    break;
  case L_Function:
    {
     result.insert(ThePoints[A] + (ThePoints[B] - ThePoints[A]) * y);
     result.insert(ThePoints[C] - (ThePoints[C] - ThePoints[B]) * y);
    }
    break;
  case S_Function:
    {
     result.insert(ThePoints[A] + (ThePoints[B] - ThePoints[A]) * y);
     result.insert(ThePoints[C]);
    }
    break;
  };
  return result;
}
```

Файл: kernel.h

```
#pragma once
#include "logic/mamdani/variablevalue.h"
#include "logic/mamdani/rule.h"
#include <QPointF>
#include <QStringList>
#include <QList>
namespace Mamdani
{
  /*
   * Класс реализующий логику алгоритма Мамдани
   */
  class Kernel
  {
  public:
    struct Result
    {
      Result()
        : Centroid(0)
      {
      }
```

```
    InputValues Input;                // Входные значения с функциями нечетких множств(термы)
    VariablePtr ResultVariable; // Выходная переменная с функциями ее термов
    QList<QStringList> Rules;   // Правила со значением подусловий
    QList<QPointF> Points;      // Ключевые точки результирующего множества
    double Centroid;            // x - координата центра тяжести фигуры описывающей результирующее множество
    QString FuzzySetName;       // имя результирующего терма
  };
 public:
  Kernel(InputValues input, RulesList rules, VariablePtr result);
 public:
  // Метод запускающий алгоритм Мамдани
  Result calculate();
 private:
  InputValues Input;                // Входные значения с функциями нечетких множств(термы)
  RulesList Rules;            // Правила, сформированные экспертами
  VariablePtr ResultVariable; // Выходная лингвистическая переменная с набором термов
 };
}
```

Файл: kernel.cpp

```
#include "kernel.h"
#include <QtDebug>
#include <QtMath>
namespace
{
 using namespace Mamdani;
 void printRule(Rule rule)
 {
  QString dbg;
  foreach(Condition condition, rule.TheConditions)
  {
   dbg.append(QString("[%1 IS %2]").arg(condition.VariableName).arg(condition.FuzzySetName));
  }
  dbg.append(QString(" then %1 IS %2").arg(rule.TheConclusion.VariableName).arg(rule.TheConclusion.FuzzySetName));
```

```
  qDebug() << dbg;
 }
 /*
  * Результат правила Мамдани
  */
 class RuleResult
 {
 public:
  RuleResult(double conditionsMin, Mamdani::FuzzySetPtr resultTerm)
   : ConditionsMin(conditionsMin)
   , ResultTerm(resultTerm)
  {
  }
  // возвращает значение результирующего усеченного предусловиями терма
  double getValue(double x)
  {
   return qMin(ResultTerm->getValue(x), ConditionsMin);
  }
  // возвращает значение предусловий
  double getConditionsMin() const
  {
   return ConditionsMin;
  }
  // возвращает список x координат пересечение результирующего терма с Y
  QSet<double> getIntersections(double y)
  {
   return ResultTerm->getIntersections(y);
  }

 private:
  double ConditionsMin;
  Mamdani::FuzzySetPtr ResultTerm;
 };
 /*
  * Итоговое множество результирующей переменной, являющееся объединением
ее термов, усеченных результатами правил.
  */
 class UnionFuzzySet
 {
```

```
 public:
  UnionFuzzySet(QList<RuleResult> rulesResults)
   : RulesResults(rulesResults)
  {
  }
  double getValue(double x)
  {
   double result = 0;
   foreach (RuleResult ruleResult, RulesResults)
   {
    result = qMax(result, ruleResult.getValue(x));
   }
   return result;
  }
  QList<double> getConditionsMinList() const
  {
   QList<double> result;
   foreach (RuleResult ruleResult, RulesResults)
   {
    result.append(ruleResult.getConditionsMin());
   }
   return result;
  }
  QSet<double> getIntersections(double y)
  {
   QSet<double> result;
   foreach (RuleResult ruleResult, RulesResults)
   {
    result.unite(ruleResult.getIntersections(y));
   }
   return result;
  }
 private:
  QList<RuleResult> RulesResults;
 };
 // расчет центра тяжести плоской фигуры ограниченной множеством точек 'p' и осью X.
 // http://www.rusnauka.com/32_DWS_2008/Informatica/36886.doc.htm формула (2)
```

```
 double getCentroid(QList<QPointF> p)
 {
  double nominatorSumma = 0;
  double denominatorSumma = 0;
  for (int i = 0; i < p.size() - 1; ++i)
  {
   nominatorSumma += 0.5 * (p[i].y() * (qPow(p[i+1].x(), 2) - qPow(p[i].x(), 2)) - 0.5 * p[i].x() * (p[i+1].x() + p[i].x()) * (p[i+1].y() - p[i].y()) + 1.0 / 3.0 * (p[i+1].y() - p[i].y()) / (p[i+1].x() - p[i].x()) * (qPow(p[i+1].x(), 3) - qPow(p[i].x(), 3)));
   denominatorSumma += 0.5 * (p[i+1].x() - p[i].x()) * (p[i+1].y() + p[i].y());
  }
  return denominatorSumma > 0 ? nominatorSumma / denominatorSumma : 0;
 }
 // расчет центра тяжести плоской фигуры результирующего нечеткого множетсва unionFuzzy
 double getCentroid(UnionFuzzySet& unionFuzzy, QList<QPointF>& points)
 {
  // ищем точки пересечения минимума каждого правила со всеми результирующими термами
  QSet<double> xSet;
  QList<double> yList = unionFuzzy.getConditionsMinList();
  foreach (double y, yList)
  {
   xSet.unite(unionFuzzy.getIntersections(y));
  }
  // сортируем точки пересечения
  QList<double> xList = xSet.toList();
  std::sort(xList.begin(), xList.end());
  // для каждой точки пересечения находим у координату лежащую на границе результирующего нечеткого множества.
  // получаем ключевые точки границы результирующего нечеткого множетсва
  foreach (double x, xList)
  {
   points.append(QPointF(x, unionFuzzy.getValue(x)));
  }
  // по ключевым точкам результирующего нечеткого множетсва находим его центр тяжести.
  return getCentroid(points);
 }
```

```
 QString getFuzzySetName(double value, QList<Mamdani::FuzzySetPtr>
outputTerms)
 {
  QString name = QObject::tr("Неопределено");
  double max = 0;
  foreach (Mamdani::FuzzySetPtr term, outputTerms)
  {
   double current = term->getValue(value);
   if (current >= max)
   {
    max = current;
    name = term->getName();
   }
  }
  return name;
 }
}
Mamdani::Kernel::Kernel(Mamdani::InputValues input, Mamdani::RulesList rules,
Mamdani::VariablePtr result)
 : Input(input)
 , Rules(rules)
 , ResultVariable(result)
{
}
Mamdani::Kernel::Result Mamdani::Kernel::calculate()
{
 Result result;
 result.Input = Input;
 result.ResultVariable = ResultVariable;
 // Обрабатываем последовательно все правила
 QList<RuleResult> rulesResults;
 foreach (Rule rule, Rules)
 {
  double conditionsMin = 1;
  QStringList verbalResults;
  bool skipRule = false;
  // Обрабатываем каждое условие правила
  foreach(Condition condition, rule.TheConditions)
  {
```

```
    if (Input.contains(condition.VariableName))
    {
      VariableValue input = Input[condition.VariableName];
      Mamdani::FuzzySetPtr term = input.TheVariable->getTerm(condition.FuzzySetName);
      double termValue = term->getValue(input.TheValue);
      conditionsMin = qMin(termValue, conditionsMin);
verbalResults.append(QString("%1[%2]").arg(condition.FuzzySetName).arg(QString::number(termValue, 'f', 2 )));
    }
    else
      skipRule = true;
  }
  skipRule = ResultVariable->getTerm(rule.TheConclusion.FuzzySetName) ? skipRule : true;
  if (!skipRule)
  {
verbalResults.append(QString("%1[%2]").arg(rule.TheConclusion.FuzzySetName).arg(QString::number(conditionsMin, 'f', 2 )));
    rulesResults.append(RuleResult(conditionsMin, ResultVariable->getTerm(rule.TheConclusion.FuzzySetName)));
  }
 }
 if (!rulesResults.empty())
 {
  UnionFuzzySet unionFuzzy(rulesResults);
  result.Centroid = getCentroid(unionFuzzy, result.Points);
  result.FuzzySetName = getFuzzySetName(result.Centroid, ResultVariable->getTerms());
 }
 return result;
}
```

Файл: rule.h

```
#pragma once
#include "logic/mamdani/condition.h"
#include "logic/mamdani/conclusion.h"
#include <QList>
namespace Mamdani
{
```

```
  /*
   * Правило для алгоритма Мамдани
   */
  struct Rule
  {
    Rule(QList<Condition> conditions, Conclusion conclusion)
      : TheConditions(conditions)
      , TheConclusion(conclusion)
    {
    }
    QList<Condition> TheConditions; // Условия правила
    Conclusion TheConclusion;       // Постусловие (заключение)
  };
  typedef QList<Rule> RulesList;
}
```

Файл: variable.h

```
#pragma once
#include "logic/mamdani/fuzzyset.h"
#include <QPair>
namespace Mamdani
{
  /*
    * Лингвистическая переменная
    */
  // Linguistic Variable {}
  // Лингвистическая переменная есть кортеж <β, T, X, G, M>, где:
  // β — имя лингвистической переменной;
  // T — множество её значений (термов);
  // X — универсум нечетких переменных;
  // G — синтаксическая процедура образования новых термов;
  // M — семантическая процедура, формирующая нечеткие множества для каждого терма данной лингвистической переменной.
  class Variable
  {
  public:
    typedef QPair<double, double> Range;
  public:
    Variable(QString name, FuzzySetList terms, Range range);
  public:
```

```
    QString getName() const;
    QList<FuzzySetPtr> getTerms() const;
    FuzzySetPtr getTerm(QString name) const;
    Range getRange() const;
  private:
    QString Name;       // β — имя лингвистической переменной;
    FuzzySetList Terms; // T — множество её значений (термов);
    Range TheRange;     // X — универсум нечетких переменных;
  };
  typedef QSharedPointer<Variable> VariablePtr;
}
```

Файл: variable.cpp

```
#include "variable.h"
Mamdani::Variable::Variable(QString name, FuzzySetList terms, Range range)
  : Name(name)
  , Terms(terms)
  , TheRange(range)
{
}
QString Mamdani::Variable::getName() const
{
  return Name;
}
Mamdani::FuzzySetList Mamdani::Variable::getTerms() const
{
  return Terms;
}
Mamdani::FuzzySetPtr Mamdani::Variable::getTerm(QString name) const
{
  foreach (FuzzySetPtr term, Terms)
  {
    if (term->getName() == name)
      return term;
  }
  return Mamdani::FuzzySetPtr();
}
Mamdani::Variable::Range Mamdani::Variable::getRange() const
{
  return TheRange;
```

```
}
```

Файл: variablevalue.h

```
#pragma once
#include "logic/mamdani/variable.h"
#include <QPair>
#include <QHash>
namespace Mamdani
{
 /*
  * Конкретное значение лингвистической переменной, введенное пользователем
  */
 struct VariableValue
 {
  VariableValue()
   : TheValue(0)
  {
  }
  VariablePtr TheVariable; // Лингвистическая переменная
  double TheValue;         // Значение переменной
  QString FuzzySetName;    // Имя терма соотв. входному значению
  QList<QPointF> Points;    // Точки результируюзего множетсва (для выходной переменной/расситывается алгоритмом)
 };
 typedef QHash<QString, VariableValue> InputValues;
}
```

2. Листинг алгоритма машины опорных векторов

Файл: svm.h

```
#pragma once
#include <vector>
#include <string>
namespace SVM
{
 /*
  * Результат SVM
  */
 enum Category
 {
```

```
    OK,
    VIRUS,
    UNKNOWN,
  };
  // реализация логики SVM
  Category Analyse(std::vector<double>& input);
  std::string toString(Category category);
}
```

Файл: svm.cpp

```
//
=================================================================
==================
//  SVM module deployment code
//  Number of variables included in the analysis = 166
//  Number of dependents (output) variables = 1
//  Number of predictors (input) variables = 165
//  Independents: Var1, Var2, Var3, Var4, Var5, Var6, Var7, Var8, Var9, Var10, Var11,
Var12,  Var13,  Var14,  Var15,  Var16,  Var17,  Var18,  Var19,  Var20,  Var21,  Var22,
Var23,  Var24,  Var25,  Var26,  Var27,  Var28,  Var29,  Var30,  Var31,  Var32,  Var33,
Var34,  Var35,  Var36,  Var37,  Var38,  Var39,  Var40,  Var41,  Var42,  Var43,  Var44,
Var45,  Var46,  Var47,  Var48,  Var49,  Var50,  Var51,  Var52,  Var53,  Var54,  Var55,
Var56,  Var57,  Var58,  Var59,  Var60,  Var61,  Var62,  Var63,  Var64,  Var65,  Var66,
Var67,  Var68,  Var69,  Var70,  Var71,  Var72,  Var73,  Var74,  Var75,  Var76,  Var77,
Var78,  Var79,  Var80,  Var81,  Var82,  Var83,  Var84,  Var85,  Var86,  Var87,  Var88,
Var89,  Var90,  Var91,  Var92,  Var93,  Var94,  Var95,  Var96,  Var97,  Var98,  Var99,
Var100, Var101, Var102, Var103, Var104, Var105, Var106, Var107, Var108, Var109,
Var110, Var111, Var112, Var113, Var114, Var115, Var116, Var117, Var118, Var119,
Var120, Var121, Var122, Var123, Var124, Var125, Var126, Var127, Var128, Var129,
Var130, Var131, Var132, Var133, Var134, Var135, Var136, Var137, Var138, Var139,
Var140, Var141, Var142, Var143, Var144, Var145, Var146, Var147, Var148, Var149,
Var150, Var151, Var152, Var153, Var154, Var155, Var156, Var157, Var158, Var159,
Var160, Var161, Var162, sendto, epoll_wait,
#include <math.h>
int noContInputs = 164;
int noCatInputs = 1;
int noInputs = 165;
int noDummyInputs = 166;
int noContOutputs = 0;
int noCatOutputs = 1;
```

```
static double dummy[166];
static double NoInputNominals[]=
{
   2
};
static double InputNominals[]=
{
   101, 102
};
int type = 0;
int ktype = 2;
double coef0 = 0;
double mgamma = 0.00606061;
double order = 0;
int noSVs = 30;

int noNominals = 2;
int noOfSVs= 30;
static int label[]=
{
   101, 102
};
static int noOfSVsPerClass[]=
{
   17, 13
};
static double SV[]=
{
   (обучающая выборка)
};
static double coefficient[]=
{
   0.329488,
   0.798311,
   0.384208,
   0.21513,
   0.203783,
   0.105673,
   0.183865,
```

```
   0.10445,
   0.283457,
   0.174576,
   0.266439,
   0.212745,
   0.0353287,
   0.652726,
   0.345414,
   0.258171,
   0.516589,
   -0.541824,
   -0.675568,
   -0.714551,
   -0.401554,
   -0.191437,
   -0.220444,
   -0.418499,
   -0.0133421,
   -0.115596,
   -0.53233,
   -0.54684,
   -0.0862378,
   -0.612129,
};
static double constants[]=
{
   -0,130972
};
/*-----------------------------------------------*/
void GetDummy(double input[])
{
   int i, j, k=0;
   for(i=0; i<noContInputs; i++) dummy[i] = input[i];
   for(i=0; i<noCatInputs; i++)
   {
      for(j=0; j<NoInputNominals[i]; j++)
      {
         if(InputNominals[k]==input[i+noContInputs]) dummy[k] = 1.0;
         else dummy[k] = 0.0;
```

```
            k++;
        }
    }
}
/*-----------------------------------------------*/
double dot(double dummy[], double sv[])
{
    int i=0, j=0;
    double sum = 0;
    while(i!=noDummyInputs && j!=noDummyInputs)
    {
        if(i==j)
        {
            sum += dummy[i]*sv[j];
            i++;
            j++;
        }
        else
        {
            if(i>j) j++;
            else i++;
        }
    }
    return sum;
}
/*-----------------------------------------------*/
double Kernel(double dummy[], int index)
{
    int i=0, j=0;
    double kernel=0, sum=0, d;
    double sv[166];
    for(i=0; i<noDummyInputs; i++) sv[i] = SV[i+index*noDummyInputs];
    switch(type)
    {
        case 0:
            kernel = dot(dummy,sv);
            break;
        case 1:
            kernel = pow(mgamma*dot(dummy, sv)+coef0, order);
```

```
   break;
case 2:
{
   while(i!=noDummyInputs && j!=noDummyInputs)
   {
      if(i==j)
      {
         d = dummy[i] - sv[j];
         sum += d*d;
         i++;
         j++;
      }
      else
      {
         if(i>j)
         {
            sum += sv[j]*sv[j];
            j++;
         }
         else
         {
            sum += dummy[i]*dummy[i];
            i++;
         }
      }
   }
   while(i!=noDummyInputs)
   {
      sum += dummy[i]*dummy[i];
      i++;
   }
   while(sv[j]!=noDummyInputs)
   {
      sum += sv[j]*sv[j];
      j++;
   }
   kernel = exp(-mgamma*sum);
}
case 3:
```

```
            kernel = tanh(mgamma*dot(dummy,sv)+ coef0);
            break;
    }
    return kernel;
}
/*-----------------------------------------------*/
double Predict(double dummy[])
{
    int i, j, k, index=0;
    double sum = 0;
    if(type==3 || type==4)
    {
        for(i=0; i<noOfSVs; i++)
        {
            sum += coefficient[i]*Kernel(dummy, i);
        }
        sum -= constants[0];
        return sum;
    }
    else
    {
        int si, sj, ci, cj, p=0;
        double *coef1, *coef2;
        int temp[2];
        int score[2];
        double kernels[30];
        for(i=0; i<noOfSVs; i++) kernels[i] = Kernel(dummy, i);
        temp[0] = 0;
        for(i=1; i<noNominals; i++) temp[i] = temp[i-1] + noOfSVsPerClass[i-1];
        for(i=0; i<noNominals; i++) score[i] = 0;
        for(i=0; i<noNominals; i++)
        {
            for(j=i+1; j<noNominals; j++)
            {
                sum = 0;
                si = temp[i];
                sj = temp[j];
                ci = noOfSVsPerClass[i];
                cj = noOfSVsPerClass[j];
```

```
                coef1 = coefficient;
                coef2 = coefficient;
                for(k=0; k<j-1; k++) coef1++;
                for(k=0; k<i; k++) coef2++;
                for(k=0; k<ci; k++) sum += coef1[si+k]*kernels[si+k];
                for(k=0; k<cj; k++) sum += coef2[sj+k]*kernels[sj+k];
                sum -= constants[p++];
                if(sum>0) ++score[i];
                else ++score[j];
            }
        }
        for(i=1; i<noNominals; i++)
        {
            if(score[i]>score[index]) index=i;
        }
        return label[index];
    }
}

/*-----------------------------------------------*/
#include <stdio.h>
#include "svm.h"
#define MISSINGCODE -9999
SVM::Category SVM::Analyse(std::vector<double>& input)
{
  Category result = SVM::UNKNOWN;
  ::GetDummy(&input[0]);
  double category = ::Predict(&input[0]);
  if (category == 101)
  {
    result = SVM::OK;
  }
  else if (category == 102)
  {
    result = SVM::VIRUS;
  }
  return result;
}
std::string SVM::toString(Category category)
```

```
{
  switch (category)
  {
    case SVM::OK:
      return "OK";
    case SVM::VIRUS:
      return "VIRUS";
    case SVM::UNKNOWN:
    default:
      break;
  }
  return "UNDEFINED";
}
```

ПРИЛОЖЕНИЕ В

Информация из журнала исследовательского прототипа модели системы обнаружения вредоносных программ

18/01/2016 23:44:37 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:37 - Генерирование вектора корректной программы...
18/01/2016 23:44:37 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:38 - Генерирование вектора корректной программы...
18/01/2016 23:44:38 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:38 - Генерирование вектора корректной программы...
18/01/2016 23:44:38 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:39 - Генерирование вектора корректной программы...
18/01/2016 23:44:39 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:39 - Генерирование вектора корректной программы...
18/01/2016 23:44:39 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:44:39 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:44:39 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:45 - Генерирование вектора корректной программы...
18/01/2016 23:44:45 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:47 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:44:48 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:44:48 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:44:49 - Генерирование вектора корректной программы...
18/01/2016 23:44:49 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:44:49 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:44:52 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[6.00%] - опасная[63.2471%]
18/01/2016 23:44:54 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:44:54 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[30.0362%]
18/01/2016 23:44:54 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[30.0362%]
18/01/2016 23:44:59 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[3.00%] - подозрительная[30.0362%]
18/01/2016 23:44:59 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[2.00%] - безопасная[17.5641%]

18/01/2016 23:44:59 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[2.00%] - безопасная[17.5641%]
18/01/2016 23:45:01 - Результат SVM[0.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[2.00%] - безопасная[17.5641%]
18/01/2016 23:45:03 - Генерирование вектора корректной программы...
18/01/2016 23:45:03 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:03 - Генерирование вектора корректной программы...
18/01/2016 23:45:03 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[2.00%] - подозрительная[43.75%]
18/01/2016 23:45:03 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[2.00%] - подозрительная[43.75%]
18/01/2016 23:45:03 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:07 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[3.00%] - подозрительная[45%]
18/01/2016 23:45:07 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[3.00%] - подозрительная[45%]
18/01/2016 23:45:08 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:09 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:09 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[6.00%] - подозрительная[45%]
18/01/2016 23:45:11 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:12 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:12 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:12 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:12 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:13 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:13 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:14 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.8246%]
18/01/2016 23:45:14 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.8246%]
18/01/2016 23:45:16 - Генерирование вектора корректной программы...
18/01/2016 23:45:16 - Результат SVM[0.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.8246%]
18/01/2016 23:45:16 - Завершено: генерирование вектора корректной программы.

18/01/2016 23:45:18 - Генерирование вектора корректной программы...
18/01/2016 23:45:18 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:21 - Результат SVM[0.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:21 - Результат SVM[0.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:21 - Результат SVM[0.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:21 - Результат SVM[0.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:45:22 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:22 - Результат SVM[0.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:22 - Результат SVM[0.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:22 - Результат SVM[0.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:23 - Результат SVM[0.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:24 - Результат SVM[0.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - безопасная[17.7778%]
18/01/2016 23:45:24 - Результат SVM[0.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - безопасная[17.7778%]
18/01/2016 23:45:25 - Генерирование вектора корректной программы...
18/01/2016 23:45:25 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:27 - Генерирование вектора корректной программы...
18/01/2016 23:45:27 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:28 - Генерирование вектора корректной программы...
18/01/2016 23:45:28 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:29 - Генерирование вектора корректной программы...
18/01/2016 23:45:29 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:30 - Генерирование вектора корректной программы...
18/01/2016 23:45:30 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:30 - Генерирование вектора корректной программы...
18/01/2016 23:45:30 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:30 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:30 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:32 - Генерирование вектора корректной программы...
18/01/2016 23:45:32 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:33 - Генерирование вектора корректной программы...

18/01/2016 23:45:33 - Результат SVM[0.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - безопасная[17.7778%]
18/01/2016 23:45:33 - Результат SVM[0.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - безопасная[17.7778%]
18/01/2016 23:45:33 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:34 - Генерирование вектора корректной программы...
18/01/2016 23:45:34 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:35 - Генерирование вектора корректной программы...
18/01/2016 23:45:35 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:36 - Генерирование вектора корректной программы...
18/01/2016 23:45:36 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:36 - Генерирование вектора корректной программы...
18/01/2016 23:45:36 - Завершено: генерирование вектора корректной программы.
18/01/2016 23:45:41 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:41 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:41 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:41 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:44 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[4.00%] - подозрительная[45%]
18/01/2016 23:45:45 - Результат SVM[1.00%] - Количество разрешений[3.00%] - Совпадение подозрительных классов[5.00%] - подозрительная[45%]
18/01/2016 23:45:47 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:47 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:48 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:48 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:49 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:49 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[5.00%] - опасная[63.2471%]
18/01/2016 23:45:51 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:51 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:52 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[6.00%] - опасная[63.2471%]
18/01/2016 23:45:53 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:53 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:54 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
18/01/2016 23:45:54 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
18/01/2016 23:45:56 - Генерирование вектора вредоносной программы...
18/01/2016 23:45:56 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:58 - Генерирование вектора вредоносной программы...

18/01/2016 23:45:58 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:45:59 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
18/01/2016 23:46:01 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:01 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:03 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:03 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:04 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
18/01/2016 23:46:05 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:06 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:06 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:06 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.8246%]
18/01/2016 23:46:06 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.8246%]
18/01/2016 23:46:07 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:07 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:10 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.8246%]
18/01/2016 23:46:14 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[7.00%] - подозрительная[45%]
18/01/2016 23:46:14 - Результат SVM[1.00%] - Количество разрешений[2.00%] - Совпадение подозрительных классов[7.00%] - подозрительная[45%]
18/01/2016 23:46:14 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:14 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:16 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:16 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:16 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:16 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:17 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:17 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:17 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:17 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:18 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[8.00%] - опасная[74.2593%]
18/01/2016 23:46:19 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[9.00%] - опасная[74.2593%]

18/01/2016 23:46:19 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[10.00%] - опасная[74.2593%]
18/01/2016 23:46:21 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:21 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:22 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:22 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:22 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:22 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:23 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[9.00%] - опасная[74.2593%]
18/01/2016 23:46:24 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[8.00%] - опасная[74.2593%]
18/01/2016 23:46:24 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[7.00%] - опасная[74.2593%]
18/01/2016 23:46:24 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
18/01/2016 23:46:24 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:46:25 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[4.00%] - опасная[74.2593%]
18/01/2016 23:46:25 - Результат SVM[1.00%] - Количество разрешений[20.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:26 - Результат SVM[1.00%] - Количество разрешений[19.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:26 - Результат SVM[1.00%] - Количество разрешений[18.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[17.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[16.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[15.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[14.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[13.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:27 - Результат SVM[1.00%] - Количество разрешений[12.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[11.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[10.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[9.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]

18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.8246%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[8.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.8246%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:28 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:29 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:29 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:30 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:30 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:31 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:31 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:32 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:32 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:33 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:33 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:34 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[3.00%] - опасная[63.2471%]
18/01/2016 23:46:34 - Результат SVM[1.00%] - Количество разрешений[4.00%] - Совпадение подозрительных классов[3.00%] - опасная[63.2471%]
18/01/2016 23:46:36 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:36 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:37 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:37 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:37 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:37 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:38 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:38 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:39 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:39 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:39 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:39 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:40 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:40 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:40 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:40 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:41 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:41 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:41 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:41 - Завершено: генерирование вектора вредоносной программы.

18/01/2016 23:46:42 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:42 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:42 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:42 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:42 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:43 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:43 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:43 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:43 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:43 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:44 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:44 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:44 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:44 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:45 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:45 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:45 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:46 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:46 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:46 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:47 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:47 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:47 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:47 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:48 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:48 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:48 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:48 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:49 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:49 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:49 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:49 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:50 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:50 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:50 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:50 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:51 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:51 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:53 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:53 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[3.00%] - опасная[74.2593%]
18/01/2016 23:46:53 - Результат SVM[1.00%] - Количество разрешений[5.00%] - Совпадение подозрительных классов[4.00%] - опасная[74.2593%]

18/01/2016 23:46:54 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:54 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:56 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[4.00%] - опасная[74.2593%]
18/01/2016 23:46:56 - Результат SVM[1.00%] - Количество разрешений[6.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:46:57 - Генерирование вектора вредоносной программы...
18/01/2016 23:46:57 - Завершено: генерирование вектора вредоносной программы.
18/01/2016 23:46:59 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[5.00%] - опасная[74.2593%]
18/01/2016 23:46:59 - Результат SVM[1.00%] - Количество разрешений[7.00%] - Совпадение подозрительных классов[6.00%] - опасная[74.2593%]
19/01/2016 00:11:45 - Генерирование вектора вредоносной программы...
19/01/2016 00:11:45 - Завершено: генерирование вектора вредоносной программы.

ПРИЛОЖЕНИЕ Г

Метод экспертный оценок для скрытой марковской модели и нечеткой когнитивной карты

Рост количества и активное развитие вредоносных программ в операционной системе Android послужили поводом исследований в сфере обнаружения вредоносных программ и разработке методики для повышения эффективности обнаружения (детектирования) вредоносных программ операционной системе данного типа. Для оценки количества времени нахождения операционной системы в одном из двух условных состояний (рабочее и нерабочее) было выполнено моделирование деятельности операционной системы Android с учетом воздействия вредоносных программ. В модели были учтены все имеющиеся в настоящее время встроенные механизмы защиты, а также поставляемые средства (антивирусы). Для получения количественных показателей необходим взгляд группы квалифицированных специалистов – экспертов в данной области. Экспертным методом необходимо выполнить оценку количества времени (в процентах) нахождения данной модели в одном из состояний, перечисленных и описанных ниже. В разработанной модели также предусмотрен переход из одного состояния в другие. Она состоит из 13 возможных состояний:

S1 – стабильное рабочее состояние ОС для мобильных устройств;

S2 – загрузка прикладных программ из известного или неизвестного источника;

S3 – прикладные программы выполняют запрос разрешений к пользователю согласно заложенному в них функционалу;

S4 – инсталлируется код полученной прикладной программы;

S5 – вредоносная программа находится в состояние покоя (ожидания);

S6 – вредоносная программа выполняет сбор информации о пользователе и устройстве;

S7 – сторонние средства защиты (антивирусная программа);

S8 – база сигнатур антивирусной программы;

S9 – встроенные средства защиты (песочница, система разрешений);

S10 – ремонт, восстановление ОС для мобильных устройств;

S11 – отказ, нерабочее состояние мобильного устройства;

S12 – вредоносная программа активируется, то есть выполняет “захват” управления над устройством;

S13 – вредоносная программа выполняет заложенный в нее функционал: пересылает смс, получает контроль и выполняет прочие вредоносные действия.

В таблице Г.1 изображена матрица переходов из одного состояния в другое. То есть первый столбец означает, что из состояния S1 модель может переходить в состояния S1, S2, S10, S11. В данной таблице изображены все возможные переходы.

Таблица Г.1

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
S1	$S_{1.1}$	$S_{1.2}$	0	0	0	0	0	0	0	$S_{1.10}$	$S_{1.11}$	0	0
S2	$S_{2.1}$	$S_{2.2}$	$S_{2.3}$	0	0	0	0	0	$S_{2.9}$	0	0	0	0
S3	0	$S_{3.2}$	$S_{3.3}$	$S_{3.4}$	0	0	0	0	$S_{3.9}$	0	0	0	0
S4	0	0	0	$S_{4.4}$	$S_{4.5}$	$S_{4.6}$	0	0	$S_{4.9}$	0	0	0	0
S5	0	0	0	0	$S_{5.5}$	$S_{5.6}$	$S_{5.7}$	0	0	0	$S_{5.11}$	$S_{5.12}$	$S_{5.13}$
S6	0	0	0	0	$S_{6.5}$	$S_{6.6}$	$S_{6.7}$	0	0	0	0	$S_{6.12}$	0
S7	0	0	0	0	0	0	$S_{7.7}$	$S_{7.8}$	$S_{7.9}$	$S_{7.10}$	0	0	0
S8	0	0	0	0	0	0	$S_{8.7}$	$S_{8.8}$	$S_{8.9}$	0	0	0	0
S9	0	0	0	0	0	0	0	$S_{9.7}$	$S_{9.9}$	0	0	0	0
S10	$S_{10.1}$	0	0	0	0	0	$S_{10.7}$	0	0	$S_{10.10}$	$S_{10.11}$	0	0
S11	$S_{11.1}$	0	0	0	$S_{11.5}$	0	0	0	0	$S_{11.10}$	$S_{11.11}$	0	0
S12	0	0	0	0	$S_{12.5}$	$S_{12.6}$	0	0	0	0	0	$S_{12.12}$	0
S13	0	0	0	0	$S_{13.5}$	0	0	0	0	0	0	$S_{13.12}$	$S_{13.13}$

В сумме каждая строка данной таблицы должна составлять 100 % (например, S1+S2+S10+S11=100%). Необходимо заполнить таблицу Г.2, в которой указать в процентах время нахождения модели в одном из описанных состояний.

Таблица Г.2

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
S1			0	0	0	0	0	0	0			0	0
S2				0	0	0	0	0		0	0	0	0
S3	0				0	0	0	0		0	0	0	0
S4	0	0	0				0	0		0	0	0	0
S5	0	0	0	0				0	0	0			
S6	0	0	0	0				0	0	0	0		0
S7	0	0	0	0	0	0					0	0	0
S8	0	0	0	0	0	0				0	0	0	0
S9	0	0	0	0	0	0	0			0	0	0	0
S10		0	0	0	0	0		0	0			0	0
S11		0	0	0		0	0	0	0			0	0
S12	0	0	0	0			0	0	0	0	0		0
S13	0	0	0	0		0	0	0	0	0	0		

Для оценки риска информационной безопасности (до и после) и эффективности внедрения системы обнаружения вредоносных программ ОС для мобильных устройств с учетом имеющихся в настоящее время средств защиты информации необходимо построить модель, на которой указывается степень влияния одних факторов на другие.

С помощью анализа, проведенного в первой главе и моделирования во второй главе, были выделены основные компоненты и условно разделены на 3 группы:

1. Ресурсы ОС для мобильных устройств, которые необходимо защищать:

- S1 – ядро ОС для мобильных устройств. Ядро является центром управления и работает с правами привилегированного пользователя;

- S2 – драйвера. Получив доступ к драйверам, можно осуществлять управление работой аппаратной части ОС для мобильных устройств;
- S3 – программное обеспечение. Управление программами как системными, так прикладными в своих целях может нанести огромный ущерб;
- S4 – система прав (разрешений). С помощью системы разрешений можно предоставить все возможные права и обеспечить достаточно ресурсов для работы вредоносных программ.

2. Перечень факторов и средств создающие определенные условия, которые могут повлечь опасность нарушения информационной безопасности ОС для мобильных устройств (см. глава 1):

- S5 – экспойт;
- S6 – неофициальная или модернизированная ОС для мобильных устройств;
- S7 – человеческий фактор;
- S8 – кража;
- S9 – вредоносные программы.

3. Виды ущерба от воздействия угроз:

- S10 – финансовые потери. В результате воздействия угроз пользователь может понести финансовые потери, например, в результате отправки платных смс, заражения банковских программ и т.д.;
- S11 – личная информация. Такая информация как номера, мультимедиа файлы, деловая переписка, документы и т.д.;
- S12 – контроль над устройством. За счет контроля над устройством можно отслеживать, например, действия пользователя, расположение устройства и т.д.

Для получения количественных показателей необходим взгляд группы квалифицированных специалистов – экспертов в данной области.

Экспертным методом необходимо выполнить оценку силы (в процентах) воздействия (влияния) одних компонентов на другие:

- 0 – не влияет;
- Low (L) – слабое влияние;
- Medium (M) – среднее влияние;
- High (H) – сильное влияние.

Необходимо заполнить таблицу Г.3 расставив степень влияния одних факторов на другие.

Таблица Г.3 – Оценка силы влияния одних факторов на другие

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
S1	0	0	0	0	0							
S2	0	0	0	0	0			0	0	0		
S3	0	0	0	0	0	0	0					
S4	0	0	0	0	0	0	0		0			
S5	0	0	0	0	0							
S6	0	0	0	0	0	0	0	0	0	0	0	
S7	0	0	0	0	0	0	0	0	0	0		
S8	0	0	0	0	0	0	0	0	0			
S9	0	0	0	0	0	0	0	0	0			
S10	0	0	0	0	0	0	0	0	0	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	0