ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
НАУКИ САНКТ-ПЕТЕРБУРГСКИЙ ИНСТИТУТ ИНФОРМАТИКИ И
АВТОМАТИЗАЦИИ РОССИЙСКОЙ АКАДЕМИИ НАУК

УДК 004.056

На правах рукописи

Браницкий Александр Александрович

ОБНАРУЖЕНИЕ АНОМАЛЬНЫХ СЕТЕВЫХ СОЕДИНЕНИЙ НА ОСНОВЕ ГИБРИДИЗАЦИИ МЕТОДОВ ВЫЧИСЛИТЕЛЬНОГО ИНТЕЛЛЕКТА

Специальность 05.13.19 —
«Методы и системы защиты информации, информационная безопасность»

Диссертация на соискание учёной степени
кандидата технических наук

Научные руководители:

доктор технических наук, профессор
Тимофеев Адиль Васильевич,

доктор технических наук, профессор
Котенко Игорь Витальевич

Санкт-Петербург — 2018

Оглавление

Введение

Актуальность темы. Разработка системы обнаружения атак (СОА) является одним из приоритетных направлений в области информационной безопасности. Важность решения этой задачи обусловливается постоянным увеличением и разнообразием компьютерных сетевых угроз, реализация которых может приводить к серьезным финансовым потерям в различных организациях. Согласно статистическим данным „Лаборатории Касперского“ в первом квартале 2017 г. было выявлено и отражено более 479 млн. компьютерных атак, в то время как за аналогичный период 2018 г. этот показатель уже превысил величину в 796 млн. атак. Подобный рост атакующих действий с каждым годом требует задействования существенно больших сил и временных затрат со стороны администраторов и аналитиков безопасности. В компаниях, вовлеченных в производство критически важной продукции, для поддержания безопасности корпоративных сетевых ресурсов расходуются крупные финансовые и материальные средства, направленные на содержание специального оборудования в виде компонентов СОА и обслуживающего его персонала. Для обеспечения корректной интерпретации передаваемых в пакетах данных необходимо выполнять их сборку в минимальный логический поток — сетевое соединение, что позволит оперировать более высокоуровневыми характеристиками сетевого трафика для выявления аномалий, свойственных сетевому и транспортному уровням модели OSI.

Для обнаружения сетевых атак могут применяться как сигнатурные механизмы поиска шаблонных аномальных действий, так и эвристические (статистические, нейросетевые, иммунные и пр.) подходы. В случае сигнатур решение задачи сводится к реализации процедуры, выполняющей проверку вхождения заданной байтовой последовательности внутри содержимого сетевых пакетов. Недостатками такого решения являются сложность создания репрезентативного набора с подобными записями и ограничение в обнаружении модифицированных вариантов известной атаки. Напротив, эвристические подходы позволяют

выявлять скрытые закономерности в анализируемых сетевых потоках. Именно эта особенность объясняет их широкую популярность в научно-исследовательском сообществе и играет ключевую роль при выборе и проектировании ядра СОА. С другой стороны, в основе функционирования большинства коммерческих и открытых программных решений преобладает подход, который базируется на сигнатурном сопоставлении с образцом и характеризуется минимальным числом ложных срабатываний. Для сохранения преимуществ обоих подходов используется прием их комбинирования, который по-прежнему остается не в полной мере исследованным. Поэтому задача обнаружения аномальных сетевых соединений является актуальной, а предлагаемый в настоящем диссертационном исследовании модельно-методический аппарат, использующий комбинирование (гибридизацию) разнородных методов вычислительного интеллекта (ВИ) и сигнатурного анализа, направлен на ее решение. Область ВИ охватывает исследование биологически инспирированных моделей (нейронных сетей, нечеткой логики, эволюционных вычислений и т.д.) [151], направленных на обработку низкоуровневых данных об объекте без использования экспертных знаний [59, 81]. Под термином «гибридизация» понимается комбинирование разнородных решателей в единую систему классификации объектов [70].

Степень разработанности темы. Вопросу обнаружения аномальных сетевых соединений посвящены работы как отечественных исследователей С.В. Безобразова, А.К. Большева, В.И. Васильева, Д.Ю. Гамаюнова, В.А. Головко, П.Д. Зегжды, И.В. Котенко, А.В. Лукацкого, О.Б. Макаревича, С.А. Петренко, В.В. Платонова, О.И. Шелухина, так и зарубежных исследователей J. Cannady, H. Debar, A.A. Ghorbani, S.A. Hofmeyr, W. Lu, V. Paxson, M. Tavallaee и др. Анализ работ в этой области показал, что для обнаружения сетевых атак отсутствует гибкая методика обучения коллектива адаптивных бинарных классификаторов, и большинство исследований ограничивается рассмотрением только одной схемы комбинирования решателей. Поэтому диссертационное исследование направлено на разработку обобщенного подхода к построению универсальной структуры для хранения и представления классификаторов — дерева классификаторов и алгорит-

ма каскадного обучения его узлов, что позволит в контексте выявления аномальных сетевых соединений объединять разнородные решатели без строгой привязки к агрегирующей их выходы композиции и повысить эффективность СОА за счет возможности выбора наилучшей схемы комбинирования решателей.

Научная задача — разработка модельно-методического аппарата для обнаружения аномальных сетевых соединений на основе гибридизации методов ВИ.

Объектом исследования являются распределенные сетевые атаки, механизмы их обнаружения и распределенные СОА.

Предметом исследования являются модели, методики и алгоритмы обнаружения аномальных сетевых соединений на основе ВИ.

Целью диссертационного исследования является повышение эффективности функционирования СОА при помощи оригинального модельно-методического аппарата, основанного на подходе «гибридизация методов ВИ». Для достижения поставленной цели решены следующие **задачи**:

1) анализ сигнатурных и эвристических методов обнаружения сетевых атак;

2) разработка программных инструментов для тестирования сетевых СОА и оценка их возможностей;

3) разработка модели искусственной иммунной системы на базе эволюционного подхода для классификации сетевых соединений;

4) разработка алгоритма генетико-конкурентного обучения сети Кохонена для обнаружения аномальных сетевых соединений;

5) разработка методики иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений;

6) разработка архитектуры и программная реализация распределенной СОА, построенной на основе гибридизации методов ВИ и сигнатурного анализа;

7) разработка программного стенда для генерации сетевых атак и экспериментальная оценка разработанной СОА.

Научная новизна диссертационного исследования заключается в следующем:

1) разработанная модель искусственной иммунной системы на базе эволюционного подхода для классификации сетевых соединений отличается от известных наличием двухуровневого алгоритма ее обучения, механизмом разрешения конфликтных случаев классификации, процедурой автоматического вычисления порога активации иммунных детекторов, а также универсальностью структуры для их представления. Для учета свойств динамического непостоянства сетевого трафика в основу функционирования данной модели заложены механизмы постоянного обновления иммунных детекторов в течение различных этапов созревания (жизненного цикла) и их переобучения с использованием расширяющегося набора аномальных сетевых записей;

2) разработанный алгоритм генетико-конкурентного обучения сети Кохонена для обнаружения аномальных сетевых соединений дополнен введением различных стратегий генетической оптимизации весовых коэффициентов «мертвых» нейронов, расположенных на выходном слое сети. Предложенная стохастическая оптимизация позволяет сократить количество эпох обучения сети Кохонена при достижении заданного максимального значения ошибки векторного квантования;

3) разработанная методика иерархической гибридизации бинарных классификаторов (детекторов) для обнаружения аномальных сетевых соединений отличается от известных возможностью задания произвольной вложенности классификаторов друг в друга и их «ленивым» подключением благодаря наличию алгоритма каскадного обучения узлов, осуществляющего эффективный нисходящий спуск по всем цепочкам зависимостей корневого классификатора. Особенность методики заключается в возможности гибкого объединения детекторов для построения единого верхнеуровневого классификатора при помощи различных низкоуровневых схем их комбинирования и агрегирующих композиций.

4) разработанная архитектура распределенной СОА, построенной на основе гибридизации методов ВИ и сигнатурного анализа, отличается от известных воз-

можностью «горячей» вставки (или замены старого) исполняемого кода, содержащего функционирование классификатора, без останова системы («на лету»), наличием интерпретатора для написания собственных сценариев, задающих структуры и правила обучения классификаторов, а также поддержкой оригинальной методики иерархической гибридизации детекторов. В отличие от других программных решений, данная СОА обладает существенно более высокой скоростью обработки сетевых потоков и более низким ресурсопотреблением.

Теоретическая и практическая значимость. Разработанные компоненты предназначены для повышения корректности детектирования сетевых атак, что позволит обеспечить необходимый уровень защищенности информационных ресурсов. Использование разработанной методики гибридизации бинарных классификаторов предоставит возможность объединить разнородные средства обнаружения сетевых атак (включая сигнатурный анализ и различные адаптивные методы) для создания гибридной СОА. За счет использования детекторов в качестве минимальной единицы классификации сетевых атак проектирование СОА ведется в стиле «снизу-вверх»: вначале ее ядро приспосабливается под выявление индивидуальных типов атак, затем оно строит правила (комбинирует детекторы, разрешает конфликты, формирует входные сигналы для классификаторов, выполняет обход дерева классификаторов) для соотнесения подозрительного соединения к тому или иному классу. Разработанный модельно-методический аппарат может быть использован как для защиты компьютерных сетей, так и для решения других более общих задач, связанных с классификацией объектов.

Методология и методы диссертационного исследования заключаются в постановке и формализации задач, связанных с обнаружением аномальных сетевых соединений, и включают методы теории множеств, теории ВИ, теории формальных языков, теории вероятностей, теории защиты информации.

Положениями, выносимыми на защиту, являются:

1) модель искусственной иммунной системы на базе эволюционного подхода для классификации сетевых соединений;

2) алгоритм генетико-конкурентного обучения сети Кохонена для обнаружения аномальных сетевых соединений;

3) методика иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений;

4) архитектура и программная реализация распределенной СОА, построенной на основе гибридизации методов ВИ и сигнатурного анализа.

Обоснованность и достоверность изложенных в диссертационной работе научных положений обеспечивается выполнением детального анализа состояния исследований в области обнаружения аномальных сетевых соединений, подтверждается согласованностью теоретических результатов с результатами, полученными при проведении экспериментов, а также публикацией в ведущих рецензируемых изданиях российского и международного уровня.

Реализация результатов работы. Представленные в диссертационной работе исследования использовались в рамках следующих научно-исследовательских работ: (1) Гранта Российского научного фонда „Управление инцидентами и противодействие целевым кибер-физическим атакам в распределенных крупномасштабных критически важных системах с учетом облачных сервисов и сетей Интернета вещей“, № 15-11-30029, 2015–2017; (2) Проекта Минобрнауки России „Разработка технологий интерактивной визуализации неформализованных данных разнородной структуры для использования в системах поддержки принятия решений при мониторинге и управлении информационной безопасностью информационно-телекоммуникационных систем“, № 14.604.21.0137, 2014–2016; (3) Проекта Минобрнауки России „Перспективные методы корреляции информации безопасности и управления инцидентами в критически важных инфраструктурах на основе конвергенции технологий обеспечения безопасности на физическом и логическом уровнях“, № 14.616.21.0028, 2014–2014. Полученные результаты внедрены в учебный процесс подготовки магистров по курсу „Advanced Network & Cloud Security“ (проект ENGENSEC TEMPUS № 544455-TEMPUS-1-2013-1-SE-TEMPUS-JPCR), используются в учебном процессе Санкт-Петербургского государственного университета телекоммуникаций

им. проф. М.А. Бонч-Бруевича и Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики.

Апробация результатов работы. Основные результаты диссертационного исследования были представлены на ряде международных и российских конференций, в том числе: 18-я IEEE международная конференция Computational Science and Engineering (Порто, Португалия, 2015), 7-я международная конференция Mathematical Methods, Models and Architectures for Computer Networks Security (Варшава, Польша, 2017), „Информационные технологии в управлении“ (Санкт-Петербург, 2012 г., 2016 г.), „СПИСОК“ (Санкт-Петербург, 2012 г.), „Методы и технические средства обеспечения безопасности информации“ (Санкт-Петербург, 2015 г., 2016 г.), „Информационная безопасность регионов России“ (Санкт-Петербург, 2015 г., 2017 г.), „РусКрипто“ (Московская область, г. Солнечногорск, 2015 г., 2018 г.), „Региональная информатика“ (Санкт-Петербург, 2016 г.), „Система распределенных ситуационных центров как основа цифровой трансформации государственного управления“ (Санкт-Петербург, 2017 г.), „Актуальные проблемы инфокоммуникаций в науке и образовании“ (Санкт-Петербург, 2018 г.) и др.

Личный вклад. Все результаты, представленные в диссертационной работе, получены лично автором в процессе выполнения научно-исследовательской деятельности.

Публикации. Основные результаты, полученные в ходе диссертационного исследования, изложены в 21 печатном издании, шесть из которых опубликованы в журналах, рекомендованных ВАК („Информационно-управляющие системы“, „Проблемы информационной безопасности. Компьютерные системы“, „Труды СПИИРАН“, „Защита Информации. Инсайд“), три — в зарубежных изданиях, индексированных в Web of Science и Scopus, 12 — в прочих изданиях. Получено три свидетельства о государственной регистрации программ для ЭВМ.

Структура и объем диссертационной работы. Диссертация состоит из введения, трех глав, заключения и пяти приложений. Основной материал из-

ложен на 204 страницах. Полный объем диссертации составляет 305 страниц с 80 рисунками и 25 таблицами. Список литературы содержит 213 наименований.

Краткое содержание работы. В **первой главе** выполнен анализ методов обнаружения сетевых атак, предложена их классификация. Определены место и роль методов ВИ в задачах ИИ и обнаружения аномальных сетевых соединений. Представлены классификация СОА и архитектура распределенной СОА, перечислены требования, предъявляемые к СОА. Выполнена постановка задачи исследования, и сформулирована цель исследования. Во **второй главе** выполнен анализ искусственных иммунных систем как одного из основных направлений ВИ, рассмотрены несколько исследоватальских прототипов СОА, построенных на основе искусственных иммунных систем. Представлены разработанные модель искусственной иммунной системы, алгоритм генетико-конкурентного обучения сети Кохонена, методика иерархической гибридизации бинарных классификаторов с приложением к обнаружению аномальных сетевых соединений. В **третьей главе** рассмотрена архитектура сетевой распределенной СОА, представлена ее программная реализация, и описаны эксперименты по оценке разработанных модели, алгоритма, методики и СОА. При проведении экспериментов использовались два набора данных — набор, созданный при помощи генератора сетевых атак, и набор данных DARPA 1998, вычислены показатели эффективности разработанной СОА с ипользованием обоих наборов данных, выполнено сравнение ее характеристик производительности с характеристиками производительности других открытых программных решений, и обосновано выполнение предъявленных к ней требований. Представлены предложения по применению разработанного модельно-методического аппарата для построения СОА.

Глава 1 Системный анализ проблемы обнаружения и классификации сетевых атак

1.1 Классификация методов обнаружения сетевых атак

По способу интерпретации входных данных методы обнаружения сетевых атак классифицируются на методы обнаружения аномалий и методы обнаружения злоупотреблений [19].

Среди первых работ, содержащих предпосылки к постановке и решению задачи обнаружения аномалий, можно считать [51] и [92]. В [51] Anderson предполагает, что злоумышленник может быть обнаружен посредством анализа содержимого журнала аудита контролируемой системы и наличием отклонений извлеченных из него записей от установленных администратором. В [92] Denning предлагает рассмотреть статистическую модель, состоящую из шести компонент: (1) субъекты (пользователь, процесс, система), (2) объекты (файлы, программы, команды, устройства), (3) записи журнала аудита, представляющие собой результат действия субъектов над объектами, (4) шаблоны поведения субъектов, (5) записи, генерируемые при обнаружении аномального поведения, и (6) правила, задающие условия их срабатывания и последующие действия.

На рисунке 1.1 представлена схема обнаружения сетевых аномалий [107].

Рисунок 1.1 — Схема обнаружения сетевых аномалий

Алгоритм обнаружения сетевых аномалий может быть описан следующим образом. Данными для анализа является сетевой трафик, представленный как набор сетевых пакетов, в общем случае фрагментированных на уровне IP. Собранные сырые данные в дальнейшем послужат источником при формировании необходимой информации для последующего анализа. Так, полученные данные могут быть агрегированы за определенный временной интервал и нормализованы с целью задания признаковых атрибутов общего вида, которые потребуются при построении текущего профиля активности. Созданный набор признаков сравнивается с набором характеристик нормальной деятельности объекта (пользователя или системы) — шаблоном нормального поведения. Если наблюдается существенное расхождение сравниваемых параметров, то фиксируется сетевая аномалия. В противном случае принимается решение о том, что данные характеристики трафика относятся к нормальному поведению. Добавление и изменение шаблонов нормального поведения может осуществляться как в ручном режиме, так и автоматически, причем некоторые параметры, задающие настройки текущего нормального профиля сетевой активности, могут изменяться в зависимости от времени суток.

Описанный выше алгоритм может включать несколько вариантов исполнения для реализации подсистемы проверки на соответствие шаблону нормального поведения. Простейшим из них является процедура сравнения с пороговой величиной, когда накопленные результаты, описывающие текущую сетевую активность, сравниваются с экспертно заданной числовой планкой. В этом подходе случай превышения значений рассматриваемых параметров указанной границы является признаком сетевой аномалии. Остальные подходы, включая этот, рассмотрены более подробно далее.

Стоит отметить, что построение шаблона нормального поведения является трудоемкой задачей и зачастую не всегда выполнимой. Так, на практике оказывается, что не каждое аномальное поведение является атакой [40]. К примеру, администратор сети может применять отладочные утилиты, такие как ping, traceroute, mtr, для диагностики сетевого окружения. Действия подобного рода

не преследуют каких-либо нелегальных умыслов, однако системы обнаружения аномалий распознают эту деятельность как свойственную нелегитимной сетевой активности.

Одной из классических и фундаментальных работ, посвященных обнаружению злоупотреблений является [142]. Для решения этой задачи авторы данной работы предлагают использовать раскрашенные сети Петри. Каждая компьютерная угроза представляется как последовательность шагов злоумышленника, которые отображаются на граф состояний системы с конечной вершиной, представленной в виде цели атакующего. Предлагаемый аппарат расширяет механизм регулярных выражений посредством введения двух условий между соседними вершинами графа — предусловия и постусловия, срабатывающих соответственно до и после шаблонного сопоставления, а также дополняет формальные грамматики поддержкой двустороннего перехода при смене состояния внутри системы. Кроме того, как отмечают авторы, предложенный ими подход позволяет объединить условные выражения и сопоставление по шаблону. Рассматриваемая раскрашенная сеть Петри имеет не более одной направленной дуги между каждым фиксированным состоянием и следующим за ним переходом, характеризуется наличием только одного конечного состояния, в то время как количество начальных состояний не ограничено. Переход в следующее состояние возможен только при условии срабатывания перехода, которое возникает в те моменты, когда (1) все входные состояния этого перехода имеют по крайней мере один маркер, и (2) системный или заданный пользователем предикат, определенный в этом переходе, принимает истинное значение. Перед выполнением перехода в новое состояние осуществляется процесс унификации связанных с маркерами переменных и значений полей сработавшего события. После этого проверяется истинность булева выражения, закрепленного за этим переходом. Достижение маркером терминального состояния равносильно обнаружению атаки, описываемой сетью Петри.

Обнаружение злоупотреблений позволяет идентифицировать несанкционированные действия, если имеется их точное представление в виде шаблонов

атак. Здесь под шаблоном атаки понимается некоторая совокупность явно описывающих конкретную атаку действий (к примеру правил сопоставления или вывода), применение которых к полям идентифицируемого объекта позволяет получить однозначный ответ о его принадлежности к этой атаке. Как и в схеме обнаружения сетевых аномалий, при обнаружении злоупотреблений в сети (рисунок 1.2 [107]) первичными данными для анализа является сетевой трафик. Выделенные атрибуты сетевых пакетов передаются в модуль, который выполняет поиск и проверку на соответствие входных данных правилам и оповещает о наличии угрозы в случае положительного срабатывания одного из этих правил. Ключевой проблемой при создании любой системы обнаружения злоупотреблений является вопрос об эффективном проектировании механизма задания правил. На практике создание исчерпывающей базы правил для выявления всевозможных атак является невозможным, поскольку описание различных вариаций атакующих действий может негативно отражаться на производительности системы. Даже несущественные изменения в атаке приводят к невозможности ее обнаружения методами на основе злоупотреблений, поэтому задаваемые правила должны быть универсальными и покрывать как можно большее число известных модификаций сетевых атак. Тем самым методы обнаружения злоупотреблений являются эффективным инструментом для выявления известных типов атак, однако их применимость по отношению к новым атакам, а также к модификациям известных атак является безрезультативной.

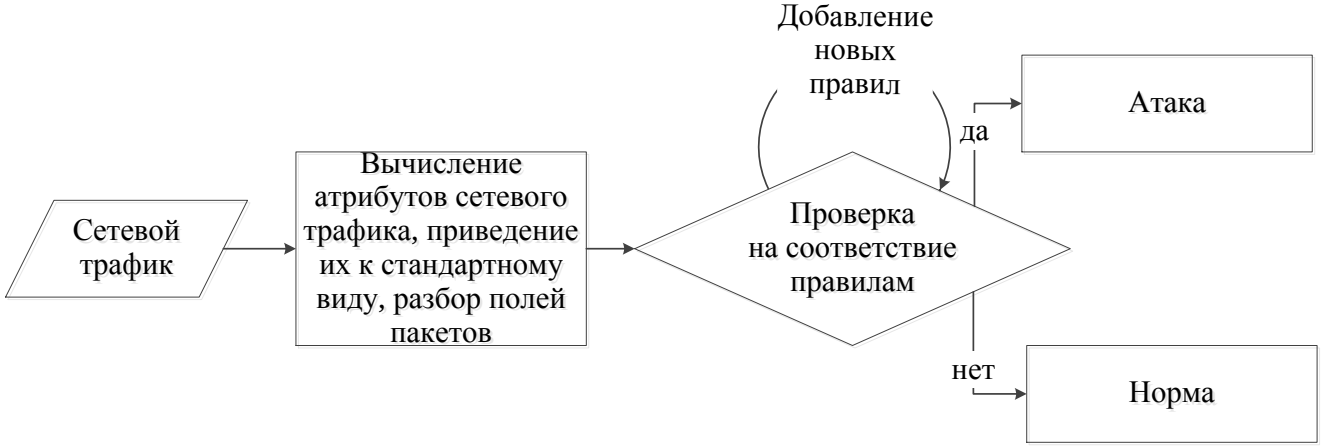


Рисунок 1.2 — Схема обнаружения злоупотреблений в сети

Классификация методов обнаружения сетевых атак, предлагаемая в настоящем диссертационном исследовании, схематически показана на рисунке 1.3.

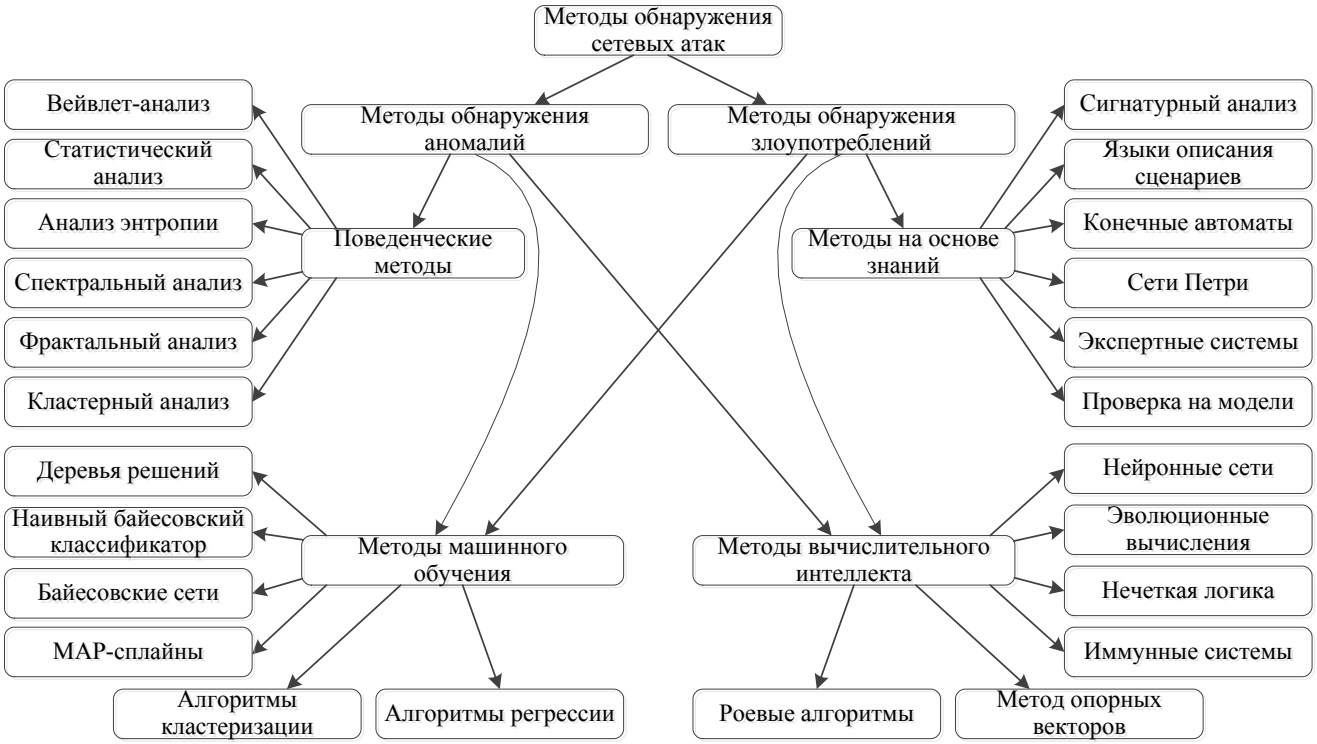


Рисунок 1.3 — Классификация методов обнаружения сетевых атак

Данная схема разбиения является условной и не претендует на полноту: некоторые из подходов могут относиться к нескольким группам. В частности, экспертные системы и конечные автоматы могут использоваться для обнаружения аномалий, но в большинстве случаев они применяются именно для обнаружения злоупотреблений. Кроме того, такие методы ВИ, как нейронные сети и метод опорных векторов, зачастую причисляют к методам машинного обучения. В схеме, соответствующей рисунку 1.3, выбрано такое разбиение, при котором методы ВИ обособляются отдельно благодаря биоподобной направленности исследуемых в рамках ВИ алгоритмов.

Поведенческими методами [91] называются методы, которые основаны на использовании информации о нормальном поведении системы и ее сравнении с параметрами наблюдаемого поведения. Представленная группа методов ориентирована на построение модели штатного, или нормального, функционирования системы или пользователя. Системы, использующие данный подход, сравнивают текущие показатели активности с профилем нормальной деятельности,

и случай значительных отклонений может рассматриваться как свидетельство наличия атаки.

В настоящем диссертационном исследовании к поведенческим методам отнесены следующие методы обнаружения сетевых атак: (1) вейвлет-анализ; (2) статистический анализ; (3) анализ энтропии; (4) спектральный анализ; (5) фрактальный анализ; (6) кластерный анализ.

Вейвлет-анализ заключается в построении коэффициентов, используемых в разложении исходного сигнала по базисным функциям. В качестве сигнала может рассматриваться интенсивность сетевого трафика [54] или данные о корреляции IP-адресов назначения [134]. Выполнение вейвлет-преобразования позволяет выделить наиболее весомую информацию как сигнал, соответствующий колебаниям с высокой амплитудой, и игнорировать менее полезную информацию в колебаниях с низкой амплитудой как шумовую составляющую.

В [55] авторы в качестве исходных данных использовали агрегированные за пятиминутные интервалы средние значения следующих величин: количество байт в секунду, количество пакетов в секунду, количество потоков в секунду, величина среднего размера TCP-пакета. В каждом случае собранные данные представляли собой дискретную последовательность частотно-временного сигнала, который согласно предложенному алгоритму вейвлет-анализа был декомпозирован в виде иерархии нескольких слоев (strata). Для каждого из извлеченных сигналов переменная времени являлась независимой. Наличие резких амплитуд в каждом из представленных сигналов соответствовало определенным группам аномалий.

Статистический анализ [92] является ядром методов обнаружения аномалий в сети. К этой группе относят следующие методы: (1) цепи Маркова; (2) метод хи-квадрат (χ^2); (3) метод среднеквадратических отклонений; (4) анализ распределений интенсивности отправки/приема пакетов; (5) анализ временных рядов; (6) пороговый анализ.

В таблице 1 приведено описание этих подходов.

Таблица 1 — Статистические методы обнаружения сетевых атак

Метод	Краткое описание метода	Преимущества метода	Недостатки метода	Пример применения
Цепи Маркова	Используется матрица перехода состояний для задания возможных связок между допустимыми действиями легитимного пользователя. Элементы матрицы — вероятности смены состояний. Событие является аномальным, если вероятность его наступления слишком мала. Вероятность наступления события сводится к вычислению произведения вероятностей отдельных составляющих этого события [67].	Возможность получения вероятности атаки. Сохранение промежуточных состояний для дальнейшего анализа вредоносных действий. Возможность применения упреждающих действий по ликвидации следующих шагов развития атаки.	Экспоненциальное увеличение числа состояний моделируемой цепи с ростом ее порядка. Необходимость в периодическом изменении параметров модели с целью ее адаптации к изменяющемуся поведению легитимного пользователя.	Обнаружение многошаговых атак. Обнаружение аномалий, которые не соответствуют заданным в цепочке правил спецификациям. Обнаружение медленных, распределенных во времени атак.
Метод χ^2	Тест по критерию χ^2 применяется для проверки того, что некоторая выборка удовлетворяет определенному закону распределения. По правилу трех сигм для нормально распределенной случайной величины, соответствующей измерениям трафика без аномалий, вычисляется верхняя граница изменения ее значения χ^2. Аномалия фиксируется в случае превышения этой границы значением χ^2 наблюдаемой величины [209].	Высокая теоретическая точность обнаружения в случае нормально распределенных случайных величин.	Предположение о нормальном распределении векторов наблюдаемых измерений, соответствующих трафику без аномалий. Необходимость задания наиболее полной выборки измерений нормального трафика.	Обнаружение сетевых аномалий, характеризующихся увеличением значений наблюдаемых параметров.
Метод среднеквадратических отклонений	На основе n предыдущих значений наблюдаемых параметров строится доверительный интервал $[m - k \cdot s, m + k \cdot s]$ для каждой из наблюдаемых величин (m — математическое ожидание, s — среднеквадратическое отклонение, k — регулируемый коэффициент, представленный как отношение z-значения к квадратному корню из объема выборки n). Событие считается аномальным, если значения его параметров не принадлежат этому интервалу.	Наличие адаптивной способности, позволяющей подстраиваться под изменения сетевого окружения.	Возможность постепенного переобучения системы со стороны злоумышленника с целью ее введения в заблуждение о нормальном поведении пользователя. Необходимость создания репрезентативной выборки нужного размера, представленной исключительно легитимным трафиком.	Обнаружение широкого класса атак, характеризующихся резкими амплитудными всплесками относительно стационарного положения.

Продолжение таблицы 1

Метод	Краткое описание метода	Преимущества метода	Недостатки метода	Пример применения
Анализ распределений интенсивности отправки/приема пакетов	Для данной сети строится модель, описывающая распределение интенсивности передачи пакетов (логнормальное, гамма и пр.). Для трафика без аномалий находятся параметры этого распределения. В тестовом режиме параметры новой построенной модели сравниваются с параметрами эталонной модели. Сетевая аномалия регистрируется в случае значительных расхождений вычисленных параметров.	Возможность наглядного визуального представления аномалии в сети с помощью графиков и гистограмм.	Зависимость качества обнаружения от выбора величины допустимого изменения параметров.	Обнаружение аномальных выбросов сетевого трафика. Обнаружение атак типа „отказ в обслуживании“ (к примеру, SYN-flooding). Выявление атак, вызывающих перегрузки сетевого оборудования.
Анализ временных рядов	Временной ряд представляет собой последовательность наблюдений, записанных в определенный момент времени [68]. Вычисленные показатели трафика соответствуют элементам дискретного ряда. Для набора этих измерений строится прогноз с использованием алгоритмов экспоненциального сглаживания, скользящего среднего или авторегрессии, позволяющих выявлять закономерности изменения трафика.	Возможность динамического и долгосрочного прогнозирования трендов, задающих изменения нормального функционирования системы. Возможность обнаружения постепенных, но значительных отклонений от нормального поведения [92].	Сложность выбора прогнозирующей функции. Усложнение описательной модели для учета сезонности наблюдаемого временного ряда. Низкая эффективность обнаружения атак для случая рядов, не обладающих свойством стационарности.	Обнаружение кратковременных фоновых атак типа „отказ в обслуживании“, приводящих к аномальным всплескам сетевой активности.
Пороговый анализ	Для наблюдаемых параметров задается допустимый диапазон изменения их значений. Нахождение вне рамок этого диапазона соответствует аномальному поведению. Простейшей модификацией, позволяющей снизить количество ложных срабатываний, является добавление счетчика, который накапливает события „выпадания“ наблюдаемых параметров из диапазона. При превышении счетчиком определенного значения фиксируется факт наличия аномалии.	Простота реализации и настройки. Отсутствие этапа предварительного обучения. Простота интерпретации полученных результатов, свидетельствующих о нормальности/аномальности событий в сети.	Необходимость тонкого задания числового порога, требующего знаний эксперта и напрямую влияющего на качество обнаружения. Отсутствие адаптивных механизмов для автоматического выбора порога. Необходимость тщательного анализа полученных результатов вследствие возможных пропусков атак и ложных срабатываний.	Обнаружение простых видов атак, направленных на подбор пароля. Обнаружение атак типа „отказ в обслуживании“ на основе сравнения интегральных характеристик трафика. Обнаружение аномалий, характеризующихся необычным временем суток для активности в сети.

Применение остальных поведенческих методов в контексте обнаружения сетевых атак рассмотрено в [5].

К методам на основе знаний относят такие методы, которые в контексте заданных фактов, правил вывода и сопоставления, отражающих признаки заданных атак, производят действия по обнаружению атак на основе заложенного механизма поиска [91].

Исследование некоторых сигнатурных механизмов поиска приведено в разделе 3.1. Примерами сигнатурных систем являются Snort и Suricata, описанные в разделах А.2 и А.3 приложения А.

Языки описания сценариев предоставляют гибкий механизм обработки контролируемых данных, который заключается в возможности написания собственных скриптов. Такой подход позволяет выявлять такие события, которые трудны для описания при помощи обычных сигнатурных инструментов анализа [130]. Ярким примером системы со встроенной поддержкой такого механизма является СОА Bro, представленная в разделе А.4 приложения А.

Метод на основе конечных автоматов (КА) предоставляет возможность моделировать атаки в виде взаимосвязанной сети из состояний и переходов. Атака считается успешно реализованной, если последовательность действий злоумышленника приводит систему из некоторого устойчивого состояния в скомпрометированное. Каждое состояние можно рассматривать как слепок параметров безопасности, а переходы между ними соответствуют успешному срабатыванию события, которое приводит систему в новое состояние. Фактически каждое наблюдаемое событие применяется к нескольким экземплярам КА, каждый из которых представляет собой определенный сценарий атаки. Одной из первых систем с поддержкой этого метода является USTAT [125].

Другим примером системы обнаружения злоупотреблений на основе анализа переходов состояний является IDIOT (Intrusion Detection In Our Time), в которой применяется метод раскрашенных сетей Петри. Сценарии вторжений кодируются в шаблоны IDIOT, и события безопасности проверяются путем их сопоставления с этими шаблонами [143].

В основе функционирования экспертных систем заложен механизм вывода, опирающийся на набор продукционных правил. В качестве примера подобной системы можно назвать IDES (Intrusion-Detection Expert System) [149], которая предназначалась для обнаружения аномалий.

Одним из наиболее популярных подходов, основанных на машинном обучении и предназначенных для обнаружения атак, является нейронная сеть. В [72] использовалась трехслойная нейронная сеть как бинарный классификатор сетевых соединений. Обучающее множество, представляющее собой сетевой трафик, полученный с помощью сетевого сканера, насчитывало около 10000 образцов соединений, среди которых 3000 записей являлись смоделированными атаками. Несмотря на то, что этап обучения занял 26.13 часа, результаты экспериментов показали высокую степень корректности распознавания.

В [53] применяются псевдобайесовские оценочные функции для определения априорных и апостериорных вероятностей новых атак. В [53] утверждается, что вследствие свойств предложенного метода системе ADAM (Audit Data Analysis and Mining) не нужны предварительные знания о шаблонах новых атак.

Существо гибридных подходов заключается в реализации различных схем объединения базовых классификаторов, которое позволяет устранять недостатки в их функционировании по отдельности. Выходные значения базовых классификаторов рассматриваются как промежуточные результаты, которые являются вспомогательными при формировании решающего правила верхнеуровневыми классификаторами [65].

В [188] представлена архитектура многоуровневой нейронной сети, в которой каждый из трех уровней представляет собой отдельный многослойный перцептрон, распределительный слой которого состоит из 30 нейронов. На каждом уровне осуществляется уточнение класса соединения. Так, на первом уровне определяется, является ли данное соединение атакой или нормальным соединением. Второй и третий уровни отвечают за классификацию соответственно класса и подкласса атаки. Особенностью данного подхода является возможность

получения необходимой степени детализации при классификации рассматриваемого соединения.

Генетические алгоритмы (ГА), как правило, применяются в совокупности с другими моделями классификации данных: деревьями решений, элементами нечеткой логики, нейронными сетями и т.д.

В [128] ГА используются для создания начальных популяций нейронных сетей. Предлагается кодировать информацию о весовых коэффициентах и архитектуре нейронной сети как двоичную последовательность. По завершении работы ГА создается группа нейронных сетей с наиболее выигрышной структурой нейронных связей, количеством скрытых слоев и начальной настройкой весов.

В [178] представлен обзор двухуровневой СОА, в которой совмещены преимущества иммунных систем и сетей Кохонена. На первом этапе происходит фильтрация признаков сетевых соединений с помощью иммунных детекторов, обученных по методу отрицательного отбора, тем самым отсеиваются те образцы, которые соответствуют нормальным соединениям. На втором этапе аномальные экземпляры обрабатываются самоорганизующимися картами и группируются в отдельные кластеры со схожими признаками. В режиме функционирования система отслеживает сетевые соединения и для каждого из них формирует вектор признаков. Поскольку первый классификатор обучен по алгоритму отрицательного отбора, то любой вектор, отличный от своих клеток, считается аномальным и подается на вход самоорганизующихся карт Кохонена. Второй классификатор отображает этот вектор на нейрон внутри кластера множества атак, имеющих общие свойства. Тем самым атаки обнаруживаются детекторами аномалий еще до проецирования на самоорганизующуюся карту.

В [211] описана двухуровневая модель, в которой для обнаружения аномалий применялись иммунные системы, а для обнаружения злоупотреблений использовались нейронные сети с предобработкой входных данных при помощи метода главных компонент.

В [43, 172] представлены три классификатора: дерево решений, метод опорных векторов и комбинация первых двух классификаторов. Функционирование третьего гибридного классификатора состояло из двух фаз. Сперва тестовые данные подавались на вход решающих деревьев, которые генерировали узловую информацию (номера листьев). Затем тестовые данные вместе с узловой информацией обрабатывались методом опорных векторов, который выдавал результат классификации. Предполагается, что при использовании этого подхода дополнительная информация, полученная от дерева решений, позволит повысить эффективность метода опорных векторов. В случае расхождения результатов классификации, полученных с помощью этих трех методов, окончательный ответ выдавался на основе взвешенного голосования.

В [160] применяется коллектив из 3 нейронных сетей и SVM. Выходное значение гибридного классификатора представляет собой взвешенную сумму выходов от четырех классификаторов, где веса вычисляются посредством минимизации среднеквадратической ошибки.

В [108, 136, 202] комбинируется аппарат иммунных систем и нейронных сетей. В качестве иммунных детекторов выбраны многослойные нейронные сети, которые генерируются при помощи метода клональной селекции.

В [110] гибридный классификатор представляется как композиция последовательно построенных на разных выборках классификаторов (машин опорных векторов и нейронных сетей с радиальными базисными функциями) и процедуры простого голосования. В результате такого объединения уровень классификации удалось повысить примерно на 1.6% по сравнению с классификаторами, взятыми по отдельности. Авторы [159] предлагают использовать выходные значения от нескольких нейронных сетей, обученных при помощи различных алгоритмов, в качестве входных значений для процедуры взвешенного голосования и голосования большинством. На тестовой выборке, состоящей из 6890 образцов, достигнута точность классификации, превышающая 99%.

В [200] описывается двухуровневая схема обнаружения сетевых атак. С этой целью несколько адаптивных нейронечетких модулей объединяются вме-

сте. Каждый из них предназначен для обнаружения только одного класса соединений. Итоговая классификация выполняется нечетким модулем принятия решений, который реализует систему нечеткого вывода Мамдани с двумя функциями принадлежности. Задачей этого модуля является определение того, насколько аномальной является обрабатываемая запись, соответствующая сетевому соединению, класс которой — это класс нечеткого модуля первого уровня с наибольшим выходным значением.

В [50] предлагается использовать k нейронных сетей с радиальными базисными функциями, причем каждая из этих сетей обучается на различных дизъюнктных подмножествах $D_1,\ldots,D_k$ исходного обучающего набора данных D. Такие подмножества генерируются при помощи метода нечеткой кластеризации, согласно которому каждый элемент $\boldsymbol{z} \in D$ относится к области D_i с некоторой степенью принадлежности $u_i^{\boldsymbol{z}}$. Каждое подмножество D_i ($i = 1,\ldots,k$) составляется из тех элементов, которые имеют наибольшую степень принадлежности к этому подмножеству среди всех остальных подмножеств. По словам авторов, за счет такого предварительного разбиения улучшается обобщающая способность классификаторов и сокращается время их обучения, поскольку для их настройки используются только те объекты, которые наиболее плотно сгруппированы вокруг образовавшегося центра обучающего подмножества. Для объединения выходных результатов $\boldsymbol{y_1},\ldots,\boldsymbol{y_k}$ этих классификаторов, принимающих на входе вектор $\boldsymbol{z}$, используется многослойная нейронная сеть, входной вектор для которой представляется в виде набора элементов, полученных в результате применения пороговой функции к каждому компоненту вектора $u_i^{\boldsymbol{z}} \cdot \boldsymbol{y_i}$ ($i = 1,\ldots,k$). Аналогичный подход был использован ранее в [203], где в роли базовых классификаторов выступали нейронные сети прямого распространения, а входом для агрегирующего их модуля, представленного также в виде классификатора указанного типа, являлись непосредственно значения векторов $u_1^{\boldsymbol{z}} \cdot \boldsymbol{y_1},\ldots,u_k^{\boldsymbol{z}} \cdot \boldsymbol{y_k}$.

В [76] для анализа записей о сетевых соединениях используются нейронечеткие модели и машины опорных векторов. Авторы этой работы выделяют четыре основных этапа в предлагаемом ими подходе. На первом этапе осуществ-

ляется генерация обучающих данных при помощи метода K-средних. Второй этап — обучение нейронечетких классификаторов. На третьем этапе выполняется формирование входного вектора для машины опорных векторов. Заключительный этап — это обнаружение атаки при помощи последнего классификатора.

В [185] для обнаружения каждого из трех типов DDoS-атак, осуществляемых с использованием протоколов TCP, UDP и ICMP, строится отдельная нейронная сеть с одним скрытым слоем. Последний слой каждой такой нейронной сети состоит из одного узла, выходное значение которого интерпретируется как наличие или отсутствие DDoS-атаки соответствующего типа. Предлагаемый подход реализован как модуль в системе обнаружения атак Snort и протестирован на трафике реального сетевого окружения.

В [44] для обнаружения DoS-атак предлагается использовать комбинированный подход, совмещающий в себе метод нормализованной энтропии и машины опорных векторов. Для выявления аномалий из сетевого трафика извлекаются шесть показателей, численно выраженных как интенсивность появления различных значений выбранных полей внутри пакетов в рамках 60-секундного окна. В данном подходе сперва вычисляются сетевые параметры при помощи метода нормализованной энтропии, затем они используются в качестве входных обучающих и тестовых данных для машины опорных векторов.

В [118] для обнаружения DoS-атак и сканирования хостов рассматривается подход, основанный на последовательном применении процедуры сжатия векторов признаков сетевых соединений и двух нечетких преобразований. Сперва ко входному восьмимерному вектору применяется метод главных компонент, уменьшающий размерность аргумента до пяти компонент с сохранением относительной суммарной дисперсии на уровне более 90%. Следующий шаг заключается в обучении или тестировании нейронечеткой сети, выходное значение которой обрабатывается при помощи метода нечеткой кластеризации.

Выполненный анализ работ показывает, что для обнаружения сетевых атак отсутствует методика, направленная на построение разнообразных коллективов классификаторов.

1.2 Место и роль методов ВИ в областях ИИ и обнаружения аномальных сетевых соединений

Согласно тесту Тьюринга [20, 29, 97, 102, 117, 201] (1950 г.) вычислительная машина является интеллектуальной в терминах искусственного интеллекта (ИИ), если она способна выполнять действия (размышления, когнитивное восприятие мира, ошибочные суждения и т.д.), подобные такому интеллектуальному существу, как человек. В рамках предложенного Тьюрингом подхода испытуемые машина и человек подвергаются диалоговой экзаменации со стороны человека-эксперта, целью которого является обнаружение сущности, не обладающей интеллектом. Этот тест требует от вычислительной техники не только способности исполнения заложенных в нее алгоритмов, но и понимания реализуемых с помощью них действий. Рассмотрим, какое место занимают методы ВИ в ИИ.

Датой зарождения понятия „вычислительный интеллект“ можно считать 1983 г. [57, 58], когда два канадских исследователя Nick Cercone и Gordon McCalla приняли решение основать научный журнал «International Journal of Computational Intelligence», первый выпуск которого датируется февралем 1985 г. Основные темы, затрагиваемые в рамках журнала, — это вопросы построения интеллектуальных агентов. В ВИ такой агент представляет собой систему, которая способна функционировать разумно (взаимодействовать с окружающей средой, вырабатывать ответные действия на основе полученных знаний и с учетом перцептуальных ограничений, адаптироваться к изменяющимся условиям и целям, обучаться на накопленных в процессе сбора данных) [81, 176, 206]. В отличие от других областей ИИ, направление ВИ в большей мере акцентировано на построении и исследовании вычислительных моделей [74]. К первым работам, в которых дается опредение ВИ, стоит отнести [59] (1992 г.) и [151] (1993 г.). В [151] Marks подчеркивает, что ВИ был введен с целью обособить от ИИ ряд теоретических дисциплин, на тот момент уже ставших вполне самостоятельными и развитыми. Среди них были выделены нейронные сети, генетиче-

ские алгоритмы, нечеткие системы, эволюционное программирование и искусственная жизнь (например, клеточные автоматы). В [57, 59] Bezdek определяет следующие характеристики, присущие вычислительно интеллектуальной системе: (1) способность обрабатывать числовые данные низкого уровня; (2) наличие компонентов распознавания образов; (3) отсутствие необходимости базироваться на экспертных знаниях. Кроме того, выделяется ряд дополнительных требований к системам подобного рода: (1) вычислительная адаптивность; (2) устойчивость в случае наличия шумов; (3) высокая скорость функционирования; (4) уровень ошибок, приближенный к человеческой деятельности.

В [95] ВИ рассматривается как „методология, включающая вычисления, которые наделяют систему возможностью обучения и/или разрешения новых ситуаций таким образом, что система воспринимается как обладающая одним или несколькими атрибутами разума, а именно обобщением, обнаружением, ассоциативностью и абстракцией“. По мнению Eberhart [81, 95], ключевую роль в методах ВИ играют их свойства адаптивности и самоорганизации, которые „делают возможным или облегчают интеллектуальное поведение объекта в сложных и изменяющихся средах“. Под адаптивностью понимается „способность системы изменять или эволюционировать свои параметры с целью лучшего выполнения задачи“, под саморганизацией — „способность системы развивать внутреннюю организацию, которая приводит к адаптированному для данной среды поведению“. Как отмечает Eberhart, важной характеристикой систем на базе ВИ является способность обобщения [95], т.е. возможность построения модели, которая достаточно точно аппроксимирует эталонную функцию по элементам, не встречавшимся в процессе настройки (обучения) системы.

Подобной точки зрения придерживается и Fogel, который отождествляет понятия адаптивности и интеллектуальности [102]: „Любая система (углеродная, кремниевая, человек, общество или особь), которая воспроизводит адаптивное поведение для достижения целей в различных средах, можно сказать, является интеллектуальной. Напротив, любая система, которая не способна воспроизводить адаптивное поведение и может работать только в одной ограниченной сре-

де, не демонстрирует никакого интеллекта.“. Кроме того, Fogel [81, 103] акцентирует направленность ВИ в виде „методов, которые могут быть использованы для адаптации решений к новым задачам без базирования на явных человеческих знаниях“, в то время как Eberhart подчеркивает биологически инспирированную сущность всех вычислительных моделей [96]. В [94] Duch отмечает, что область ВИ не ограничивается лишь разработкой интеллектуальных агентов, она также охватывает и исследование тех задач, для которых отсутствует явный алгоритм их решения. Это утверждение раскрывает ту мысль, что области ИИ и ВИ сохраняют соприкасающиеся понятия и имеют схожие проблемы. Здесь, как и в области ИИ, среди таких задач по-прежнему актуальными остаются следующие: (1) задачи распознавания зрительных образов (вычисление расстояния до объекта в 3D-пространстве, управление транспортным средством); (2) задачи, связанные с обеспечением человеко-машинного взаимодействия на основе естественного языка (генерация смысловых предложений, обработка речевых запросов); (3) творческие задачи (написание стихов и музыкальных композиций, воспроизведение эмоциональной речи); (4) задачи, связанные с перебором большого числа вариантов (игра в шахматы, доказательство теорем) и пр.

Итак, ВИ является подмножеством ИИ [59], и можно утверждать, что всякая вычислительно интеллектуальная система является интеллектуальной в терминах ИИ, но обратное неверно [132]. Для того, чтобы понять существенные отличия ВИ от ИИ рассмотрим коммутативную диаграмму ABC, предложенную в [59]. В ней Bezdek выделяет три уровня абстракции и сложности для интеллектуальных систем (A — системы в смысле ИИ, B — системы в смысле биологического интеллекта, C — системы в смысле ВИ). В отличие от ВИ, область ИИ охватывает более трудоемкие задачи, исходные данные для которых необходимо не только обрабатывать в виде численных сигналов, но также и интерпретировать с привлечением логических методов, позволяющих вычленить их семантику. Для преодоления барьера между C и A, как утверждает Bezdek, первая система должна обладать „способностью связывать низкоуровневые данные с правилами, фактами и ограничениями с целью увеличения понимания

системой окружающей ее среды“ [59]. Тем самым ИИ требует наличия некоторых дополнительных знаний (knowledge tidbits) [57, 59, 132], которые позволяют осуществить переход от численной обработки данных об объекте к пониманию его структуры и окружения и сформировать базис для структурных отношений между объектами на символическом уровне [115]. Существенным различием, отделяющим В от А, является наличие высокоскоростной ассоциативной (автоассоциативной и гетероассоциативной [20]) памяти, которая позволяет человеку генерировать с каждым сохранившимся в памяти образом огромный поток связанных с ним воспоминаний, или ассоциаций. Можно сказать, что в попытке приблизиться хоть как-то к устранению этого пробела и создана концепция глубокого обучения, в которой оперируемые данные принадлежат различным уровням абстракции.

Отличия между областями ИИ и ВИ заключаются в следующем:

- В то время как ВИ обрабатывает числовое представление информации, ИИ рассматривает данные в символическом виде [206];
- ВИ анализирует структуру объекта в стиле „снизу вверх“, опираясь на низкоуровневую информацию о нем, а ИИ основывается на построении системы в стиле „сверху вниз“, полагаясь на высокоабстрактное представление об объекте [206];
- Методы ВИ направлены на моделирование естественных процессов или систем, связанных с интеллектуальным поведением [81], а ИИ ориентирован непосредственно на моделирование интеллектуального, или человеческого, поведения (восприятие, рассуждения, обучение, общение, выработка действий в сложной среде, а также самосовершенствование [60, 132]) посредством извлечения человеческих знаний [137, 163];
- Если традиционные системы ИИ используют явные экспертные знания и точный логический вывод, что может приводить к увеличению объема хранимых данных и неоптимальному поиску, то ВИ приоткрывает завесу над понятием эвристических подходов, способных осуществлять приближенный поиск оптимального решения за меньшее число шагов [137].

Рассмотрим применение теории эволюционных вычислений и конечных автоматов (КА) к обнаружению аномальных сетевых соединений. Суть использования КА заключается в построении такой модели, которая при срабатывании внешнего или внутреннего условия осуществляет переход в одно или несколько возможных состояний. На практике чаще всего рассматривают детерминированные КА (ДКА), характеризующиеся однозначностью перехода при смене состояний. Определим ДКА как пятерку $DFA = \langle Q,\Sigma,\delta,q_0,F\rangle$ [22]:

- Q — множество состояний ДКА (конечное);
- Σ — множество входных символов (входной алфавит);
- $\delta : Q \times \Sigma \to Q$ — функция изменения состояния;
- q_0 — начальное состояние ($q_0 \in Q$);
- F — множество конечных состояний ($F \subset Q$).

Данный математический аппарат хорошо подходит для описания поведения динамических объектов. На основе имеющейся априорной информации о текущем состоянии объекта и внешнем воздействии (условии перехода) можно спрогнозировать и смоделировать его следующее состояние. Возможны два способа генерации КА с использованием эволюционных вычислений:

- генетические алгоритмы (решение представляется в виде последовательности битовых символов);
- генетическое программирование (решение представляется в виде дерева выражений).

Рассмотрим первый подход, который включает применение генетических алгоритмов для создания КА, на примере обнаружения трех типов атак: сканирования, DoS-атаки и атаки, характеризующейся большим количеством неудачных попыток удаленного входа в компрометируемую систему. На рисунке 1.4 показан пример представления простого КА, являющегося одним из решений задачи об обнаружении этих классов атак. Начальным состоянием является состояние $q_0 = S_0$, множество конечных состояний состоит из четырех элементов $F = \{S_0, S_5, S_6, S_7\}$[1].

[1]Начальное состояние включено в множество конечных состояний для удобства описания непрерывного процесса обнаружения атак.

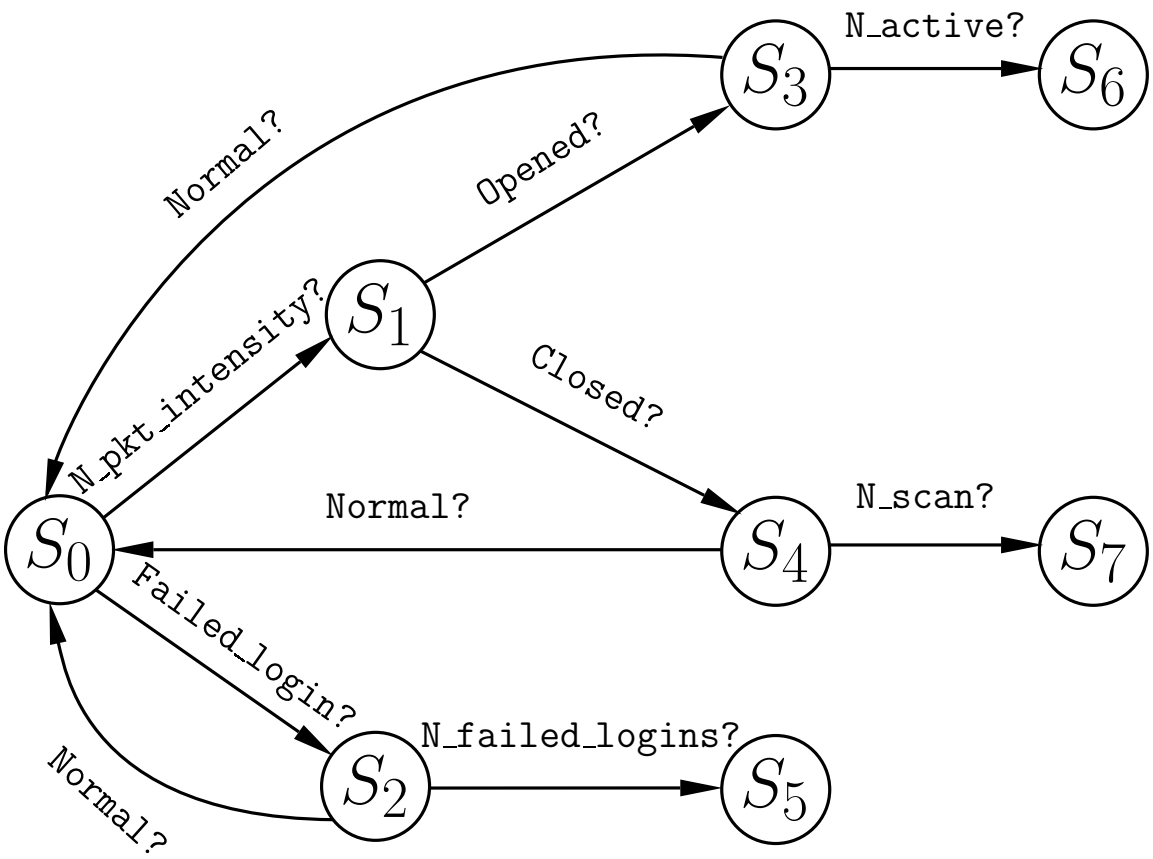


Рисунок 1.4 — Пример КА, распознающего класс атаки

В таблице 2 показано табличное представление данного КА.

Таблица 2 — Табличное представление КА

Текущее состояние		Входное условие		Переходное состояние	
S_0	000	`N_pkt_intensity?`	000	S_1	001
S_0	000	`Failed_login?`	001	S_2	010
S_1	001	`Opened?`	010	S_3	011
S_1	001	`Closed?`	011	S_4	100
S_2	010	`N_failed_logins?`	100	S_5	101
S_2	010	`Normal?`	101	S_0	000
S_3	011	`N_active?`	110	S_6	110
S_3	011	`Normal?`	101	S_0	000
S_4	100	`N_scan?`	111	S_7	111
S_4	100	`Normal?`	101	S_0	000

В таблице 2 каждому состоянию и условию сопоставлена в соответствие цепочка из трех двоичных чисел $[01]\{3\}$. В качестве условных переходов использовались следующие показатели:

- `N_pkt_intensity?` — булев показатель того, является ли высокой интенсивность входящих сетевых пакетов (т.е. превышающей величину N входящих сетевых пакетов за определенный временной интервал);
- `Failed_login?` — булев показатель того, является ли неуспешной данная аутентификация;
- `Opened?` — булев показатель того, является ли соответствующее TCP-соединение открытым в данный момент;
- `Closed?` — булев показатель того, является ли соответствующее TCP-соединение закрытым в данный момент;
- `N_failed_logins?` — булев показатель того, превышает ли число допустимых неверных попыток аутентификации на N за определенный временной интервал;
- `N_active?` — булев показатель того, превышает ли количество активных в данный момент соединений указанной величины N;
- `N_scan?` — булев показатель того, превышает ли число сканирующих пакетов за единицу времени указанной величины N;
- `Normal?` — булев показатель, логически эквивалентный $(not\,$`N_active?`$)\ and\ (not\,$`N_scan?`$)\ and\ (not\,$`N_failed_logins?`$)$.

Заметим, что в рассматриваемом примере во внимание принимаются только TCP-пакеты. Кроме того, данный КА уже приведен к стандартному виду (осуществлена нумерация вершин в соответствии с обходом в ширину), который позволяет однозначно представить его в виде последовательности битовых символов.

Другие алгоритмы представления КА в канонизированном виде описаны в книге [75].

В данном примере КА кодируется в виде хромосомы, содержащей следующий набор генов:

$$000\ 001000\ 010001\ 011010\ 100011\ 101100\ 000101\ 110110\ 000101\ 111111\ 000101.$$

Первая последовательность из трех битов (000) соответствует начальному состоянию КА S_0. Следующие последовательности из 6 битов — это пары вида (переходное состояние, входное условие). Стоит отметить, что у сгенерированных таким образом КА все состояния, кроме конечных, должны иметь ровно два условных перехода.

На начальной стадии генетического алгоритма создается популяция из нескольких экземпляров КА. Это поколение может быть создано либо произвольно, либо на основании какого-либо оптимизационного критерия. В качестве такого критерия может выступать предварительный отбор некоторого количества КА, которые имеют качество распознавания не меньше указанной величины. Значение функции приспособленности позволяет оценить вклад каждого КА в решение данной задачи. Отбираются те экземпляры, которые имеют наибольшее значение функции приспособленности.

В результате применения оператора скрещивания двух решений создается новое решение, возможно имеющее большее значение функции приспособленности по сравнению с первоначальными решениями. Для этого два предка обмениваются случайным образом выбранными участками хромосом. Оператор мутации подразумевает произвольное изменение некоторых генов внутри хромосомы экземпляра. Новая популяция формируется частично из старого генофонда и новых представителей с наибольшим значением функции приспособленности.

Алгоритм завершается, если превышено количество итераций его выполнения либо сформированная группа решений имеет среднеквадратическую ошибку, не превышающую указанного порога.

Другим подходом к генерации КА является генетическое программирование [138]. В этом случае функционирование КА предлагается описывать в виде дерева выражений. На рисунке 1.5 показано такое представление.

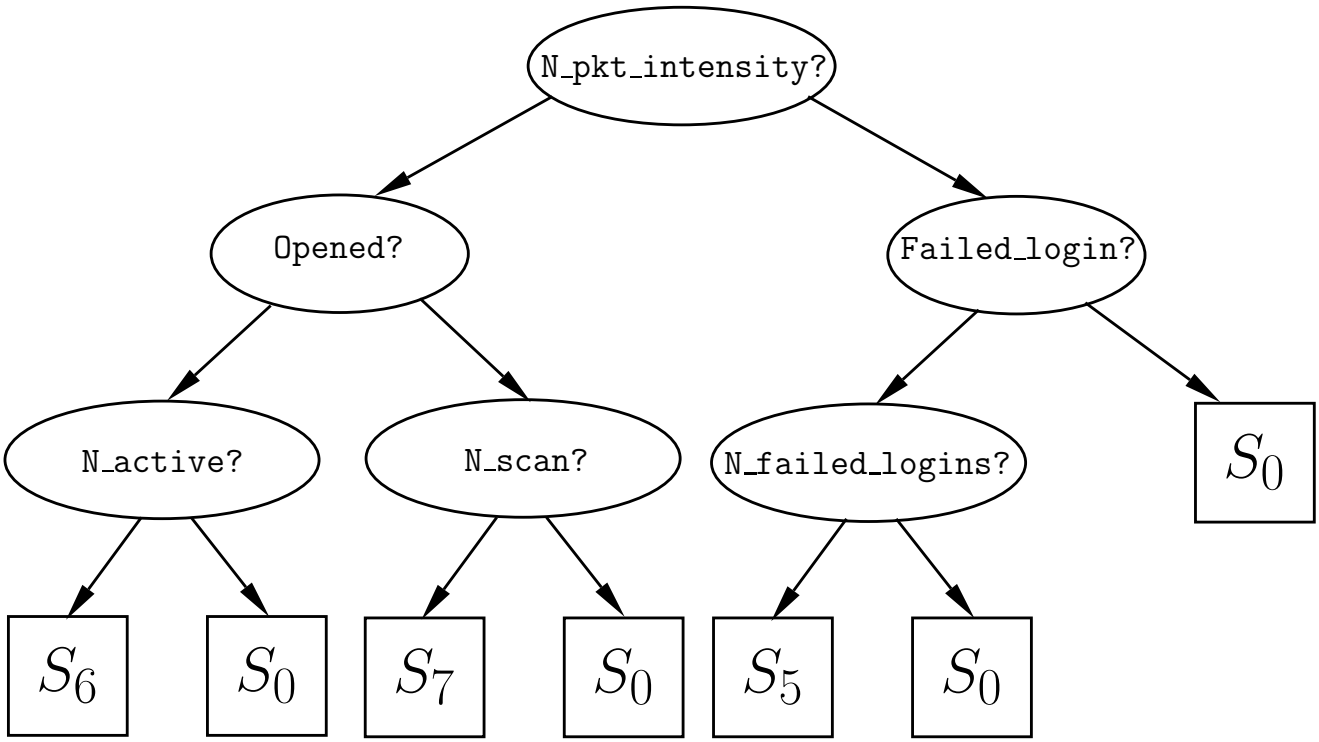


Рисунок 1.5 — Представление КА в виде дерева выражений

В роли нетерминальных узлов в таком дереве выступают операторы условных переходов. Переход в левую ветвь поддерева выполняется в том случае, если истинно значение в узле-предке, иначе выполняется переход в правое поддерево. Множеством терминальных узлов (листьев) является множество конечных состояний рассматриваемого КА, при достижении которых аномальное соединение классифицируется в соответствии с одним из них.

Алгоритм формирования новых популяций аналогичен описанному выше генетическому алгоритму. Отличием является лишь то, что операторы скрещивания и мутации приводят к обмену и изменению поддеревьев, а не битовых подстрок, как в случае генетических алгоритмов.

Рассмотрим несколько проблем, возникающих в процессе решения задачи обнаружения сетевых аномалий при помощи методов машинного обучения (и в частности ВИ) и предложим несколько рекомендаций, которые позволят разработчикам СОА задуматься о целесообразности применения таких подходов.

На данный момент машинное обучение применяется с большим успехом во многих областях: оптическом распознавании символов, обнаружении спама, идентификации биометрических показателей, построении рекомендательных

сервисов. Но в то же время при использовании этой же технологии возникает ряд трудностей с обнаружением аномалий в компьютерных сетях. *С чем может быть связана особенность использования методов машинного обучения применительно к задаче обнаружения сетевых аномалий и как объяснить природу такого отличия от решения других задач на базе этой же технологии?* По определению обнаружение аномалий включает в себя детектирование ранее не встречавшихся атак, в то время как методы машинного обучения в контексте постановки такой задачи направлены всего лишь именно на поиск взаимосвязей и закономерностей в сетевом трафике, нахождении активности, похожей на ранее встречавшуюся в обучающей выборке [193]. Строго говоря, это не совсем одно и то же, что многие привыкли считать как способность методов машинного обучения к обнаружению новых типов атак. Применение инструментов машинного обучения в готовом виде „из коробки“ к задаче обнаружения аномалий приводит к большому числу ложных срабатываний и пропусков атак. В первую очередь это обусловлено тем, что сетевой трафик постоянно меняется изо дня в день. Кроме того, трудно отследить цикличность или сезонность такого изменчивого поведения. Поэтому одним из подходов к решению этой проблемы с использованием методов машинного обучения является динамическая подстройка интеллектуальных детекторов к изменяющимся условиям. Но все равно в коммерческих реализациях систем обнаружения атак главным образом используются методы на основе правил, с помощью которых можно вручную более тонко отслеживать и задавать варьирование параметров в наблюдаемой сети. Это объяснимо рядом их сильных сторон, среди которых можно назвать возможность обоснования, почему был зафиксирован тот или иной сигнал атаки в конкретный момент времени, а также возможность более простой настройки системы. На основании имеющегося набора правил и характеристик трафика можно выявить причину срабатывания этого правила. При использовании средств машинного обучения трудно сделать подобные выводы по отношению к детекторам аномалий. Во-первых, сам по себе такой детектор вследствие сложного алгоритма обучения и особенностей своего внутреннего устройства представляет собой

черный ящик, функционирование которого скрыто от пользователя и разработчика. Во-вторых, если детектор с течением времени постоянно меняется, то это также усложняет задачу объяснения того факта, что сигнал тревоги, полученный от детектора, действительно является признаком атаки. Для упрощения этой задачи, возможно, потребуется дополнительно вводить процедуру сохранения промежуточных слепков, отражающих состояния детектора в фиксированные срезы времени, чтобы иметь возможность просмотра истории изменения его состояний с откатом к предыдущим слепкам. В противном случае, если детектор постоянно остается статичным, то это приводит к увеличению показателей ложных срабатываний и пропуска атак в будущем.

Следующей проблемой в области обнаружения аномалий является задача определения того, что же все-таки является нормальным трафиком, а что нет. Ведь если надо обнаруживать аномалии, то необходимо определить фильтр нормальности, чтобы максимально полно описать нормальную деятельность, откидывать все ненормальные действия и обучать модели только на нормальных данных. *Как же отличать случаи, когда действительно легитимный трафик просто меняется, а когда начинается деятельность, свойственная атакующему?* Можно предположить, что смена характера трафика сопровождается резким и мощным всплеском сетевой активности от конкретного узла или подсети и существенными отклонениями наблюдаемых статистических параметров. Но здесь возникает обратная сторона такого отчасти ошибочного и поспешного рассуждения: могут возникать редкие варианты, когда при малых изменениях значений отслеживаемых признаков все равно проводится своеобразная атака. К примеру, атакующий может пытаться постепенно обучать детекторы машинного обучения, очень медленно приспосабливая их к все более вредоносной активности (т.н. „эффект кипяченой лягушки“ [77, 78][2]). После такой подстройки новые детекторы будут принимать ранее аномальные образцы как нормальные. Во избежание таких случаев нужно четко понимать, как отслеживаемые дан-

[2]Данное явление получило свое название благодаря тому факту, что лягушка, помещенная в емкость с горячей водой, сразу же выпрыгнет из нее, в то время как лягушка, сперва помещенная в холодную, но постепенно доводящуюся до кипения воду, не заметит изменения температуры воды и медленно сварится заживо.

ные меняются с течением времени. Но, к сожалению, это, как правило, трудно осуществимо на практике. Поэтому можно обойтись тем требованием, чтобы детекторы расширяли свою область инспектирования очень медленно, кроме того, полезно задавать некоторые пороговые значения, дальше которых „расширяться“ детекторам не дозволено.

Наконец, последний и, пожалуй, один из самых важных вопросов — *как выбирать признаки, характерные для нормального трафика и аномалий и пригодные для обучения моделей машинного обучения?* Как один из вариантов для решения этого вопроса необходимо опытным путем сперва попробовать строить временные последовательности с наиболее полным набором признаков, характеризующих наблюдаемые явления в сети, и далее на основе экспериментов отсекать все инвариантные свойства, которые сохраняются при смене типа трафика, как заведомо неинформативные. После этого обучение будет проводиться на наборе оставшихся атрибутов. Другой способ построения признаков заключается в добавлении некоторой автоматизации к описанному выше подходу: сначала в максимально возможном объеме описывается множество всех контролируемых атрибутов, а потом применяются методы корреляционного анализа для устранения компонент, а иногда и их линейных комбинаций, с близкой к нулю дисперсией. Оставшийся набор признаков может быть использован для обучения и проверки модели машинного обучения.

Итак, *какое применение методов машинного обучения видится к задаче обнаружения аномалий?* (1) Можно использовать эти методы для задания пороговых значений. (2) Можно использовать их для преобразования (предобработки) входных параметров, к примеру при помощи следующих методов машинного обучения: метода главных компонент (PCA, principal component analysis), сингулярного разложения (SVD, singular value decomposition), внешне не связанных уравнений (SUR, seemingly unrelated regressions). (3) Можно использовать их совместно с сигнатурными методами на основе правил.

1.3 Классификация СОА и архитектура распределенной СОА

СОА — программное или аппаратное средство, которое обеспечивает ряд функций: обеспечение безопасности подлежащих защите информационных ресурсов, выявление подозрительных и аномальных событий безопасности, обнаружение фактов неавторизованного доступа в компьютерную систему или сеть. Основная задача СОА заключается в обеспечении обнаружения и предотвращения угроз безопасности.

СОА по типу ответной реакции делятся на пассивные и активные. Первые направлены только на обнаружение факта наличия злонамеренной деятельности в системе или сети, в то время как активные СОА также способны противодействовать и блокировать атаки.

По типу обрабатываемой информации можно выделить хостовые и сетевые СОА. Хостовые СОА обеспечивают сбор касающейся данного хоста информации о различных событиях безопасности. К таким событиям можно отнести вызов определенных системных команд, запуск программ или служб, изменение конфигурационных файлов. Эти СОА характеризуются способностью обнаруживать атаки, направленные на компрометацию именно того ПО и оборудования, которое установлено на защищаемом хосте. Сетевые СОА обрабатывают информационные потоки, полученные от сетевых датчиков, и обнаруживают сетевые аномалии.

По типу архитектуры СОА можно разделить на два класса: автономные и клиент-серверные [19]. Принцип работы автономных СОА основан на том, что сбор информации и анализ трафика и журналов регистрации событий осуществляются на одном компьютерном узле. Поэтому подобные системы отличаются быстрой скоростью работы и требуют больших вычислительных ресурсов. Второй класс СОА, также называемых распределенными, является наиболее распространенным. Популярность использования таких систем обусловливается несколькими факторами: модульностью, централизованным доступом и возмож-

ностью анализа трафика, циркулирующим между всеми наблюдаемыми хостами. Правильное размещение компонентов мониторинга трафика в этих системах позволяет получать всеобъемлющую картину угроз, возможных для данной компьютерной сети.

Среди основных компонентов распределенной СОА можно перечислить следующие:

– сенсоры (агенты сбора первичной информации);
– коллектор;
– модуль выявления вредоносной активности;
– подсистема реагирования на возникновение инцидентов безопасности;
– консоль администрирования;
– внешние хранилища данных;
– база правил.

На рисунке 1.6 схематически изображены СОА и ее основные компоненты.

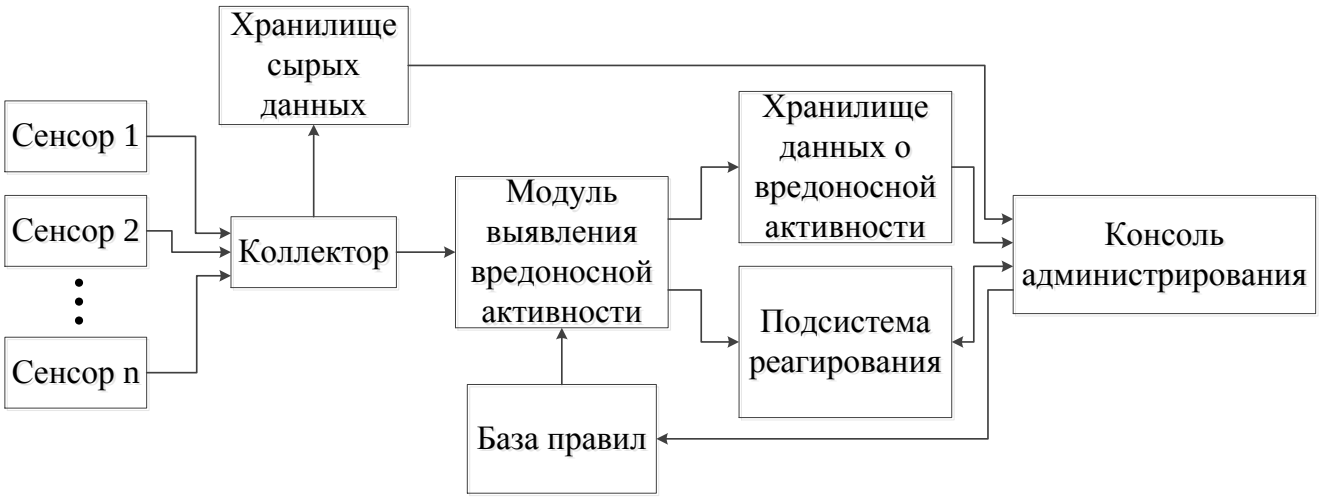


Рисунок 1.6 — Архитектура распределенной СОА

Рассмотрим каждый из компонентов подробней. Сенсоры представляют собой датчики, которые устанавливаются в контролируемые сегменты компьютерной сети или на определенные хосты. Чаще всего они реализуются как анализаторы сетевых пакетов и журналов событий или модули в ядре ОС, СУБД, ФС. Задачей сенсоров является сбор данных, фильтрация (с целью удаления избыточных и бесполезных данных), нормализация (с целью приведения данных к

общему виду для передачи их следующему модулю) и первичное обнаружение явных нарушений безопасности (например, сигнатурный анализ). Коллектор, в свою очередь, принимает на вход данные, полученные от нескольких сенсоров, дополнительно фильтрует входящий поток, выполняет процедуру корреляции с целью последующего выявления распределенных атак и помещает их в хранилище сырых данных.

Модуль выявления вредоносной активности может быть реализован как компонент обнаружения аномалий или злоупотреблений. В первом случае данные, подвергающиеся анализу, сравниваются с шаблонами профилей нормальной деятельности. Обнаружение злоупотреблений основано на сопоставлении текущих характеристик событий безопасности шаблонам (сигнатурам) атак. База правил, используемая этим модулем, либо задается вручную экспертом, либо настраивается путем обучения системы.

Подсистема реагирования в случае обнаружения подозрительных событий и сетевых соединений должна производить одно из следующих действий:

– оповещать администратора безопасности;
– сбрасывать активное соединение;
– блокировать входящие соединения с определенным IP-адресом.

Если событие помечается как подозрительное, его атрибуты помещаются в хранилище данных о вредоносной активности для возможности последующего анализа со стороны администратора.

Консоль администрирования представляет собой отдельно установленный компьютер или виртуальный хост, который содержит в себе функции по удаленному конфигурированию и доступу к отдельным компонентам системы. В частности, должна быть предоставлена возможность просмотра инцидентов безопасности, для которых был сгенерирован сигнал тревоги, и сырых данных для более детального изучения причин срабатывания модуля обнаружения атак.

1.4 Требования, предъявляемые к СОА

Выделим следующий набор функциональных требований для СОА [28]:

1. Возможность анализа заголовков сетевых пакетов.

СОА должна обрабатывать значения полей внутри пакетов, которые нарушают спецификации RFC или назначение которых не регламентировано в требованиях как поддерживаемое (например, такие значения могут быть зарезервированы для будущего использования).

2. Возможность анализа содержимого отдельных сетевых пакетов, дефрагментированных последовательностей IP-пакетов и TCP-потоков.

В СОА должен быть заложен механизм анализа содержимого как отдельных и дефрагментированных пакетов, так и реассемблированных TCP-сессий.

3. Возможность сборки фрагментированного трафика.

В СОА должны быть встроены функции обработки фрагментированных пакетов и их корректной склейки.

4. Возможность обнаружения скрытых атак, атак со вставкой [23, 180].

СОА должна игнорировать пакеты с неправильной контрольной суммой, пакеты с неподдерживаемой версией протокола IP (к примеру, в некоторых организациях используется для обмена по сетям исключительно IPv4, и любые другие пакеты могут рассматриваться как подозрительные), IP-пакеты со значением поля TTL, меньшим, чем число хопов до указанного в пакете адреса получателя. Кроме того, СОА должна обнаруживать сегменты, левая или правая граница которых перекрывает границы соседних сегментов.

5. Возможность обработки пакетов, нарушающих стандартное поведение TCP-сессии.

СОА должна корректно обрабатывать повторные запросы на установление соединения в рамках существующего соединения, пакеты с отсутствующим флагом квитирования и синхронизации порядковых номеров, а также любые другие отклонения от легальных переходов в машине состояний TCP-стека.

В качестве нефункциональных требований для СОА выделим следующие:

1. Своевременность.

Под своевременностью понимается способность системы выполнять заданную задачу в пределах установленного временного интервала. В режиме обучения максимальные временные ограничения для настройки адаптивных модулей СОА — 300 минут. В режиме анализа сетевых потоков данных скорость обработки пакетов — 300 килопакетов (300×1024 пакетов) в секунду. Такие показания выбраны на основе стандартных требований, предъявляемых к системам подобного вида.

2. Обоснованность.

Под обоснованностью понимается способность системы выдавать результаты, соответствующие реальному классу анализируемого объекта. Для разрабатываемой системы показатель корректности классификации сетевых соединений устанавливается на уровне 95%. Такая величина выбрана на основе анализа работ [43, 72, 118, 172, 178, 188, 200] в исследуемой области. Другие исследуемые показатели, которые вводятся для оценки эффективности функционирования СОА, представлены в разделе 2.5.

3. Ресурсопотребление.

Под ресурсопотреблением понимается набор и количество вычислительных ресурсов, выделяемых системе для решения заданной задачи. Поскольку часть из доступных ресурсов может быть занята сторонними процессами (процессом инициализации, оконным менеджером, сетевыми службами, консолью управления), для разрабатываемой системы вводится ограничение 50%, или не более 1024 МВ ОЗУ и не более половины задействованных в решении задачи ядер от их общего числа.

1.5 Постановка задачи исследования

Задача классификации объектов в классической постановке описывается следующим образом [11]. Дано множество всевозможных объектов — $\mathcal{X}$ и его подмножество, содержащее объекты, подлежащие анализу в рамках рассматриваемой задачи, — $\mathcal{X}_\mathcal{C}$. Данные множества не обязательно являются конечными или даже счетными. Также задано множество классов $\mathcal{C} = \{C_0, \ldots, C_m\}$, каждый элемент которого представляет собой набор объектов, содержащих внутренние представления с неизвестной взаимосвязью, и является подмножеством множества $\mathcal{X}_\mathcal{C}$: $\forall C \in \mathcal{C}\ C \subset \mathcal{X}_\mathcal{C}$[3].

Каждому входному объекту $\boldsymbol{z} \in \mathcal{X}_\mathcal{C}$ соответствует набор описывающих его атрибутов $(z_1, \ldots, z_n)^T$ — вектор признаков. В дальнейшем понятия „объект“, „экземпляр“, „запись“, „образ“ и „вектор признаков“ будут отождествляться и использоваться как взаимозаменяемые.

Не умаляя общности, можно считать, что каждому объекту $\boldsymbol{z}$ в соответствие сопоставлен ровно один класс, и множество $\mathcal{X}_\mathcal{C}$ является дизъюнктным объединением множеств, описывающих отдельные классы $C_i\ (i = 0, \ldots m)$, а именно $\mathcal{X}_\mathcal{C} = \bigvee_{i=0}^{m} C_i$. В противном случае имеется ввиду, что решается задача мультиклассификации, которая должна быть сведена к предыдущей задаче, к примеру с помощью введения в объекты дополнительных признаков, позволяющих однозначно характеризовывать входные объекты (и их принадлежность классам). Требуется построить классификационный алгоритм (правило) $R' \in \mathcal{R} = \{\mathcal{X}_\mathcal{C} \longrightarrow \{0, \ldots, m\}\}$, который минимизирует ошибку неверного соотнесения объектов тому или иному классу (функционал эмпирического риска):

$$\boxed{\Omega\left(R', \mathcal{X}_\mathcal{C}\right) = \frac{1}{\#\mathcal{X}_\mathcal{C}} \cdot \#\left\{\boldsymbol{z} \,|(\exists C \in \mathcal{C}\ \boldsymbol{z} \in C) \wedge R'\left(\boldsymbol{z}\right) \neq \zeta\left(C\right)\right\}_{\boldsymbol{z} \in \mathcal{X}_\mathcal{C}} \longrightarrow \min_{R \in \mathcal{R}}.}$$

[3] Во избежание различного рода противоречий в наивной теории множеств понятия „класс“ и „множество“ аксиоматически не являются одним и тем же [26], однако в данном диссертационном исследовании эти понятия используются в более узком смысле и в большей мере как „совокупность объектов“.

Здесь для удобства описания классификационного алгоритма введено вспомогательное отображение $\zeta : \mathcal{C} \longrightarrow \{0,\ldots,m\}$, которое сопоставляет каждому классу его номер (метку): $\forall C_i \in \mathcal{C}\ \zeta(C_i) = i\ \ (i = 0,\ldots,m)$. На практике $\mathcal{X} = \mathbb{R}^n$ и $\#\mathcal{X}_{\mathcal{C}}$ конечная. Рассматриваемые классификаторы являются адаптивными с точки зрения их способности к настройке внутренних параметров в процессе постоянного предъявления на их вход векторных выборок. С другой стороны, классификаторы обладают отличными друг от друга количественными и качественными характеристиками (размерность входного и выходного слоев, процесс преобразования анализируемых векторов, значения влияющих на настройку алгоритма параметров и т.д.), что позволяет говорить об их гетерогенности.

Задача, поставленная в данном диссертационном исследовании, может быть сформулирована следующим образом. Даны базовые классификаторы $F^{(1)},\ldots,F^{(P)} : \mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$, обученные на наборе маркированных векторов признаков сетевых соединений $\Upsilon_{\mathcal{X}_{\mathcal{C}}^{(LS)}} = \{(\boldsymbol{x_i},\bar{c}_i)\}_{i=1}^{M} = \bigcup_{C\in\mathcal{C}} \bigcup_{\boldsymbol{z}\in C\bigcap\mathcal{X}_{\mathcal{C}}^{(LS)}} (\boldsymbol{z},\zeta(C))$ ($\boldsymbol{x_i}\in C_{\bar{c}_i}$ — отношение принадлежности), и агрегирующая функция коллектива классификационных правил F: $\left(2^{\{0,\ldots,m\}}\right)^P \times\mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$, которая комбинирует выходные результаты классификаторов $F^{(1)},\ldots,F^{(P)}$. Здесь $\mathcal{X}_{\mathcal{C}}^{(LS)}\subseteq\mathcal{X}_{\mathcal{C}}$ — множество обучающих векторов. Каждое из представленных классификационных правил $F^{(1)},\ldots,F^{(P)},F$ в качестве выходного значения формирует множество $\{i_j\}_{j=0}^{m'} \subseteq \{0,\ldots,m\}\ \ (0\leqslant m'\leqslant m)$, чьи элементы обозначают возможные метки классов с точки зрения этого классификатора. Кроме того, сама функция F может представлять собой сложную многоуровневую процедуру, что затрудняет разработку общего подхода для построения классификационного правила на основе коллектива решателей $F^{(1)},\ldots,F^{(P)}$. Требуется произвести настройку агрегирующей функции F таким образом, чтобы функционал эмпирического риска ее композиции $G = F\circ\left[F^{(1)},\ldots,F^{(P)},\mathbf{id}\right] : \mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$ с базовыми классификаторами $F^{(1)},\ldots,F^{(P)}$ не превышал минимального значения функционала эмпирического риска отдельных классификационных правил:

$$\boxed{\Omega\left(F\circ\left[F^{(1)},\ldots,F^{(P)},\mathbf{id}\right],\mathcal{X}_{\mathcal{C}}\right) \leqslant \min_{j\in\{1,\ldots,P\}} \Omega\left(F^{(j)},\mathcal{X}_{\mathcal{C}}\right).}$$

Подход, именуемый в данном диссертационном исследовании как „гибридизация методов ВИ“, подразумевает объединение именно разнородных решателей, уже зафиксированных после их надлежащей настройки, при помощи разнообразных агрегирующих функций и рассматривается в контексте решения задачи обнаружения и классификации аномальных сетевых соединений. С этой целью были экспериментально исследованы следующие агрегирующие композиции: (1) мажоритарное голосование, (2) взвешенное голосование, (3) многоярусная укладка, (4) метод Фикса—Ходжеса.

Мажоритарное голосование $\overset{(\mathrm{MV})}{G}$ [144, 212] задается следующим образом:

$$\overset{(\mathrm{MV})}{G}(\boldsymbol{z}) = \left\{ k \,\middle|\, \sum_{j=1}^{P} \left[k \in F^{(j)}(\boldsymbol{z}) \right] > \frac{1}{2} \cdot \sum_{i=0}^{m} \sum_{j=1}^{P} \left[i \in F^{(j)}(\boldsymbol{z}) \right] \right\}_{k=0}^{m}. \tag{1.1}$$

В результате применения функции $\overset{(\mathrm{MV})}{G}$ к анализируемому объекту $\boldsymbol{z}$ образуется множество меток тех классов, за которые было отдано больше половины общей суммы голосов, полученных от базовых классификаторов $F^{(1)},\ldots,F^{(P)}$. Расширением этой функции является взвешенное голосование $\overset{(\mathrm{WV})}{G}$, представимое в следующей форме:

$$\overset{(\mathrm{WV})}{G}(\boldsymbol{z}) = \mathrm{Arg} \max_{i \in \{0,\ldots,m\}} \sum_{j=1}^{P} \omega_j \cdot \left[i \in F^{(j)}(\boldsymbol{z}) \right]. \tag{1.2}$$

Коэффициенты ω_j подбираются таким образом, чтобы выполнялось следующее тождество: $\sum\limits_{j=1}^{P} \omega_j = 1$. К примеру, для их вычисления можно использовать следующую формулу:

$$\omega_j = \frac{\#\left\{ \boldsymbol{x_k} \,\middle|\, \bar{c}_k \in F^{(j)}(\boldsymbol{x_k}) \right\}_{k=1}^{M}}{\sum_{i=1}^{P} \#\left\{ \boldsymbol{x_k} \,\middle|\, \bar{c}_k \in F^{(i)}(\boldsymbol{x_k}) \right\}_{k=1}^{M}}.$$

В этой формуле каждый коэффициент ω_j ($j = 0,\ldots,P$) — это доля верно (но, возможно, неоднозначно) распознанных классификатором $F^{(j)}$ обучающих векторов среди всех остальных обучающих векторов, верно распознанных коллективом решателей $F^{(1)},\ldots,F^{(P)}$. При подстановке значений $\omega_j = \frac{1}{P}$ в формулу 1.2 получаем ее частный аналог — простое голосование (голосование max-wins).

Многоярусная укладка $\overset{(\mathrm{ST})}{G}$ [25, 179] представляет собой композицию некоторой функции $\overset{(\mathrm{ST})}{F}$ с базовыми классификаторами $F^{(1)},\ldots,F^{(P)}$ и тождественной функцией $\mathbf{id}$ и задается при помощи следующей формулы:

$$\overset{(\mathrm{ST})}{G}(\boldsymbol{z}) = \overset{(\mathrm{ST})}{F}\left(F^{(1)}(\boldsymbol{z}),\ldots,F^{(P)}(\boldsymbol{z}),\boldsymbol{z}\right). \tag{1.3}$$

В формуле 1.3 функция $\overset{(\mathrm{ST})}{F}$ агрегирует входной вектор $\boldsymbol{z}$ и возвращаемые функциями $F^{(1)},\ldots,F^{(P)}$ результаты над вектором $\boldsymbol{z}$. Метод Фикса—Ходжеса [30, 101] основан на идее формирования у каждого классификатора области компетентности, в пределах которой верхнеуровневый классификатор $\overset{(\mathrm{FH})}{G}$ ему полностью „доверяет“. Выбор классификатором $\overset{(\mathrm{FH})}{G}$ решения, полученного от того или иного базового классификатора $F^{(l)}$ ($l=1,\ldots,P$) над входным объектом $\boldsymbol{z}$, обусловлен тем, насколько корректно распознает отобранный базовый классификатор $\widetilde{M}\leqslant M$ обучающих экземпляров, лежащих в наиближайшей окрестности от вектора $\boldsymbol{z}$. Тем самым функция $\overset{(\mathrm{FH})}{G}$ может быть задана следующим образом:

$$\overset{(\mathrm{FH})}{G}(\boldsymbol{z}) = \bigcup_{l=1}^{P}\left(F^{(l)}(\boldsymbol{z})\left|\underbrace{\sum_{j=1}^{\widetilde{M}}\left[F^{(l)}(\boldsymbol{x}_{\boldsymbol{i}_j}) = \left\{\bar{c}_{i_j}\right\}\right]}_{D_{\widetilde{M}}(l)} = \max_{k\in\{1,\ldots,P\}} D_{\widetilde{M}}(k)\right.\right), \tag{1.4}$$

$$\left\{i_1,\ldots,i_{\widetilde{M}}\right\} = \arg\min_{\substack{\widetilde{I}\subseteq\{1,\ldots,M\}\\(\#\widetilde{I}=\widetilde{M})}} \sum_{i\in\widetilde{I}} \rho\left(\boldsymbol{x}_{\boldsymbol{i}},\boldsymbol{z}\right).$$

В формуле 1.4 объединяются метки тех классов, за которые проголосовали классификаторы, имеющие наибольшее количество верно распознанных обучающих объектов, расстояние от которых до вектора $\boldsymbol{z}$ минимально согласно метрике ρ. С этой целью вычисляется индексное подмножество $\left\{i_1,\ldots,i_{\widetilde{M}}\right\}$ обучающей выборки $\mathcal{X}_{\mathcal{C}}^{(LS)}$, векторы с номерами которого удовлетворяют заданному требованию. Модификациями [30] являются следующие формулы:

$$\overset{(\mathrm{FH})}{G}(\boldsymbol{z}) = \bigcup_{l=1}^{P}\left(F^{(l)}(\boldsymbol{z})\left|\underbrace{\sum_{j=1}^{\widetilde{M}}\frac{\left[F^{(l)}(\boldsymbol{x}_{\boldsymbol{i}_j}) = \left\{\bar{c}_{i_j}\right\}\right]}{\rho\left(\boldsymbol{x}_{\boldsymbol{i}_j},\boldsymbol{z}\right)}}_{D^{*}_{\widetilde{M}}(l)} = \max_{k\in\{1,\ldots,P\}} D^{*}_{\widetilde{M}}(k)\right.\right), \tag{1.5}$$

$$\overset{\text{(FH)}}{G}(\boldsymbol{z}) = \bigcup_{l=1}^{P} \left(F^{(l)}(\boldsymbol{z}) \left| \underbrace{\sum_{\substack{j \in \{1,\ldots,M\} \\ \rho(\boldsymbol{x_j},\boldsymbol{z}) \leqslant \widetilde{r}}} \left[F^{(l)}(\boldsymbol{x_j}) = \{\bar{c}_j\} \right]}_{D^{**}_{\widetilde{r}}(l)} = \max_{k \in \{1,\ldots,P\}} D^{**}_{\widetilde{r}}(k) \right. \right). \qquad (1.6)$$

В формуле 1.5 выполняется взвешенное поощрение классификаторов, прямо пропорциональное близости верно классифицированного обучающего объекта и тестового объекта $\boldsymbol{z}$. В формуле 1.6 осуществляется суммирование не по всем ближайшим $\widetilde{M}$ обучающим экземплярам, а только по экземплярам, лежащим в окрестности с заданным радиусом $\widetilde{r}$ от тестового объекта $\boldsymbol{z}$.

Здесь и в дальнейшем на протяжении всей оставшейся части диссертации неизменными и недвусмысленными останутся следующие обозначения:

- $\mathcal{C} = \{C_0, \ldots, C_m\}$ — множество классов;
- $\zeta : \mathcal{C} \longrightarrow \{0, \ldots, m\}$ — отображение, сопоставляющее классу его метку;
- $\mathcal{X}_{\mathcal{C}}^{(LS)} = \{\boldsymbol{x_i}\}_{i=1}^{M}$ — обучающее множество;
- $\Upsilon_{\mathcal{X}_{\mathcal{C}}^{(LS)}} = \{(\boldsymbol{x_i}, \bar{c}_i)\}_{i=1}^{M}$ — маркированная обучающая выборка;
- $\Upsilon_{\mathcal{X}_{\mathcal{C}}^{(TS)}} = \{(\boldsymbol{z_i}, \bar{c}_i)\}_{i=1}^{M^*}$ — маркированная контрольная выборка;
- $\boldsymbol{z} \in \mathcal{X}_{\mathcal{C}}^{(TS)}$ — тестовый объект контрольной выборки;
- n — размерность входных векторов признаков сетевых соединений;
- $m + 1$ — мощность множества классов ($\#\mathcal{C}$):
  - 0 — метка класса нормальных сетевых соединений;
  - $\{1, \ldots, m\}$ — множество меток аномальных сетевых соединений;
- M — мощность обучающей выборки ($\#\mathcal{X}_{\mathcal{C}}^{(LS)}$);
- M^* — мощность контрольной (тестовой) выборки ($\#\mathcal{X}_{\mathcal{C}}^{(TS)}$);
- $F^{(1)}, \ldots, F^{(P)} : \mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$ — коллектив базовых классификаторов;
- $F : \left(2^{\{0,\ldots,m\}}\right)^P \times \mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$ — агрегирующая функция;
- $G : \mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}$ — агрегирующая композиция;
- P — порядок коллектива классификационных правил [30];
- $\Omega : \left\{\mathbb{R}^n \longrightarrow 2^{\{0,\ldots,m\}}\right\} \times \mathbb{R}^n \longrightarrow \mathbb{R}$ — функционал эмпирического риска.

Выводы по главе 1

В главе 1 решены следующие задачи:

1. Выполнен анализ подходов к обнаружению аномалий и злоупотреблений в компьютерных сетях. В рамках этого анализа предложена классификация методов обнаружения сетевых атак.
2. Показаны место и роль методов ВИ в областях ИИ и обнаружения аномальных сетевых соединений. Предложен ряд рекомендаций для использования методов машинного обучения при обнаружении сетевых аномалий.
3. Выполнена постановка задачи диссертационного исследования, которая заключается в разработке модельно-методического аппарата для обнаружения аномальных сетевых соединений на основе гибридизации методов ВИ.

Глава 2 Методы ВИ для обнаружения и классификации аномальных сетевых соединений

2.1 Естественная иммунная система и модели искусственных иммунных систем

Постоянная эволюция живых организмов привела к образованию особого защитного биологического механизма, способного функционировать на клеточно-молекулярном уровне и успешно справляться со множеством неблагоприятных для данного организма внешних факторов, — естественной иммунной системы. Среди ее отличительных особенностей можно выделить способность распознавания любых молекул вне зависимости от их происхождения и принадлежности к собственным тканям организма или чужеродным микроорганизмам (патогенам) [196], наличие ассоциативной памяти, способствующей идентификации не только одной определенной формы патогена, но и структурно связанных с ней других шаблонных вариаций [88, 192], отсутствие централизованного контроля, обеспечивающее распределенное поведение компонент иммунной системы без необходимости взаимодействия с каким-либо единым управляющим органом [84], предоставление многоуровневой линии обороны с использованием физического (слизистые оболочки, эпителиальные клетки Лангерганса), физиологического (ферменты, щелочно-кислотные и температурные среды) и лейкоцитарного (клетки врожденной и адаптивной иммунных систем) барьеров [120].

Иммунная система представляет собой совокупность клеток и молекул, скоординированная деятельность которых заключается в осуществлении главных функций — поддержания стабильной работы (гомеостаза) и защиты структурной целостности организма от множества патогенных микроорганизмов (вирусов, бактерий, паразитов). При попадании в него чужеродного тела происходит активация защитных белковых молекул, и усиливается активная ответная реакция, которая заключается в выработке особых клеток, способных напрямую

противодействовать или осуществлять косвенную помощь в нейтрализации этого тела. В то же время сама иммунная система обладает сложными саморегулирующими процессами, которые заключаются в ингибировании активности порожденных ею клеток.

Основными клетками, задействованными в выполнении функций иммунной системы, являются лейкоциты, а именно фагоциты и лимфоциты [27], располагающиеся в различных структурных компонентах живого организма. Роль фагоцитов заключается в поглощении вредоносных микроорганизмов и поедании остатков мертвых тканей. Считается, что данный процесс, именуемый фагоцитозом, вместе с активацией фагоцитов составляет первичную линию защиты в виде врожденной иммунной системы [37, 41]. Наиболее ярко выраженной по агрессии и быстроте функцией фагоцитоза обладает особый класс фагоцитов, представленных нейтрофилами, которые прибывают первыми на место очага инфицированного воспаления и при контакте с зараженной клеткой поглощают ее, после чего мгновенно подвергаются запрограммированной нормальной клеточной гибели (апоптозу). Моноциты, также относящиеся к фагоцитам, помимо данной функции, активно секретируют сигнальные белковые молекулы, именуемые цитокинами, которые направлены на привлечение к иммунному ответу других медиаторов иммунной системы. Другие представители этого же семейства, а именно макрофаги, являющиеся образами созревших и дифференцировавших моноцитов, выполняют роль антигенпрезентирующих клеток (АПК). Макрофаги способны сохраняться в организме в течение нескольких месяцев или лет, в то время как нейтрофилы циркулируют в крови и тканях в течение нескольких суток и после выполнения своих функций фагоцитируются макрофагами [3]. Лимфоциты, представленные В-клетками и Т-клетками, являются основными участниками вторичной линии защиты — адаптивной иммунной системы.

Принцип работы В-клеток заключается в распознавании антигенов (чужеродных молекул, вызывающих иммунный ответ) посредством рецепторов (прообразов антител), находящихся на поверхности самой В-клетки. В свою очередь, каждый антиген имеет на своей поверхности дискретные участки, называемые

эпитопами. При обнаружении антигена В-клетка взаимодействует с ним на основе мозаичного принципа: ее рецепторы комплементарно связываются с соответствующими эпитопами антигена, блокируя его деятельность. В-клетки являются результатом развития гемопоэтических стволовых клеток, которые можно наблюдать в красном костном мозге или селезенке млекопитающих [27]. Начальное количество таких клеток до первого инфицирования невелико. Однако одной из мер защиты, обеспечиваемых такими клетками, является выработка репертуара подобных им клеток за счет механизма клональной экспансии, увеличивающей скорость элиминации антигена, и сохранение наиболее приспособленных из них в виде пула клеток памяти. Одновременно с этим процессом антигенные рецепторы на поверхности В-клетки подвергаются соматической гипермутации, приводящей к улучшению связывания между В-клеткой и определенным антигеном. Именно благодаря этим процессам у млекопитающих усиливается иммунологический ответ: при повторной встрече с одним и тем же антигеном организм способен подавить его атаку более быстро и эффективно за счет наличия в организме достаточного числа уже сформировавшихся нужных антител различной формы. Стоит отметить, что каждая такая клетка способна продуцировать идентичные по форме антитела и может обнаруживать только ограниченный набор сходных между собой антигенов. Отсюда становится очевидным: для того чтобы успешно выполнять защитные функции, иммунная система должна состоять из большого числа различных клеток. В то же время за счет процесса клонирования клетки, имеющие одинаковые рецепторные молекулы и стимулируемые одним и тем же антигеном, организуются в группы (или популяции). По некоторым подсчетам число различных таких групп составляет величину, не меньшую 1 млн. [27], что свидетельствует о том, что иммунная система обладает сложным кооперативным процессом обнаружения и устранения вредоносных патогенных микроорганизмов.

Т-клетки, как и В-клетки, создаются в костном мозге, но дополнительно проходят этап окончательного формирования в тимусе. Именно в этом органе осуществляется процедура отрицательного отбора, которая позволяет исключить

клетки, способные реагировать против собственных клеток организма. Выделяют несколько классов Т-клеток. К первому классу относятся клетки, выполняющие вспомогательные функции в процессе иммунного ответа (Т-хелперы). В частности, после генерации зрелых В-клеток должна быть выполнена их активация с использованием антигена и соответствующей хелперной Т-клетки. Рецепторы новообразовавшейся В-клетки связываются с антигеном, что одновременно сопровождается выработкой сигнала от хелперной Т-клетки, которая заставляет В-клетку продуцировать антитела. В условиях отсутствия такого сигнала последняя клетка анергирует (деактивируется), что приводит к возможной ее гибели [31]. Второй класс клеток — это Т-супрессоры, роль которых заключается в подавлении иммунного ответа. Данные клетки позволяют регулировать размер популяции Т-хелперов. Третий класс Т-клеток включает Т-киллеры, известные также как цитотоксические Т-клетки. При их функционировании применяется механизм, позволяющий разрушать те клетки, на поверхности которых находятся фрагменты разрушенных антигенов. Сперва чужеродный патоген захватывается АПК, которая выполняет функцию фагоцитоза. После ее попадания в лимфоузлы эта клетка предъявляет пептид соответствующего антигена Т-лимфоцитам, и как следствие сама клетка-носитель распознается Т-клеточными рецепторами как инородное тело, подлежащее дальнейшему уничтожению. Эта способность является одной из основных функций цитотоксических Т-клеток и называется лизисом. Каждая Т-клетка такого класса способна выполнять такой процесс несколько раз, при этом не подвергаясь саморазрушению. Между всеми этими Т-клетками существует тесная связь. Так, Т-хелперы при обнаружении патогенного микроорганизма выделяют особые ферменты (лимфокины), которые принадлежат подклассу цитокинов и приводят к размножению и созреванию Т-киллеров, непосредственно участвующих в уничтожении этого патогена. В то же время Т-супрессоры вызывают эффект затухания порождаемых Т-хелперами и Т-киллерами иммунных реакций, который необходим для предотвращения развития аутоиммунных заболеваний.

Особенностью В-клеток является их способность напрямую связываться с эпитопами антигенов без использования сторонних участников иммунного ответа, в то время как для Т-клеток характерно распознавание антигенных частиц, предварительно обработанных другими клетками.

Механизм адаптивной иммунной системы, опосредуемый Т-клетками, носит название клеточного иммунного ответа и состоит из следующих представленных на рисунке 2.1 фаз [14, 37, 84, 89]:

1. фагоцитоз антигена АПК;
2. фрагментация переваренного антигена на отдельные пептиды и предъявление одного из них вместе с молекулой главного комплекса гистосовместимости (ГКГ) в виде комбинации пептид-ГКГ на поверхности АПК;
3. активация хелперных Т-клеток посредством связывания их рецепторных молекул с комбинацией пептид-ГКГ;
4. выработка лимфокинов активированными хелперными Т-клетками;
5. стимуляция В-клеток и киллерных Т-клеток к пролиферации при помощи секретированных Т-хелперами лимфокинов;
6. разрушение киллерными Т-клетками наружной мембраны зараженной клетки;
7. подавление развития хелперных и киллерных Т-клеток через выделяемые супрессорными Т-клетками лимфокины.

Механизм адаптивной иммунной системы, опосредуемый В-клетками, носит название гуморального иммунного ответа и состоит из следующих представленных на рисунке 2.2 фаз [14, 37, 84, 89]:

1. связывание антигенного рецептора В-клетки с эпитопом антигена;
2. секреция лимфокинов Т-хелперами;
3. клонирование В-клетки и формирование плазматических клеток и клеток памяти;
4. продуцирование антител плазматическими клетками.

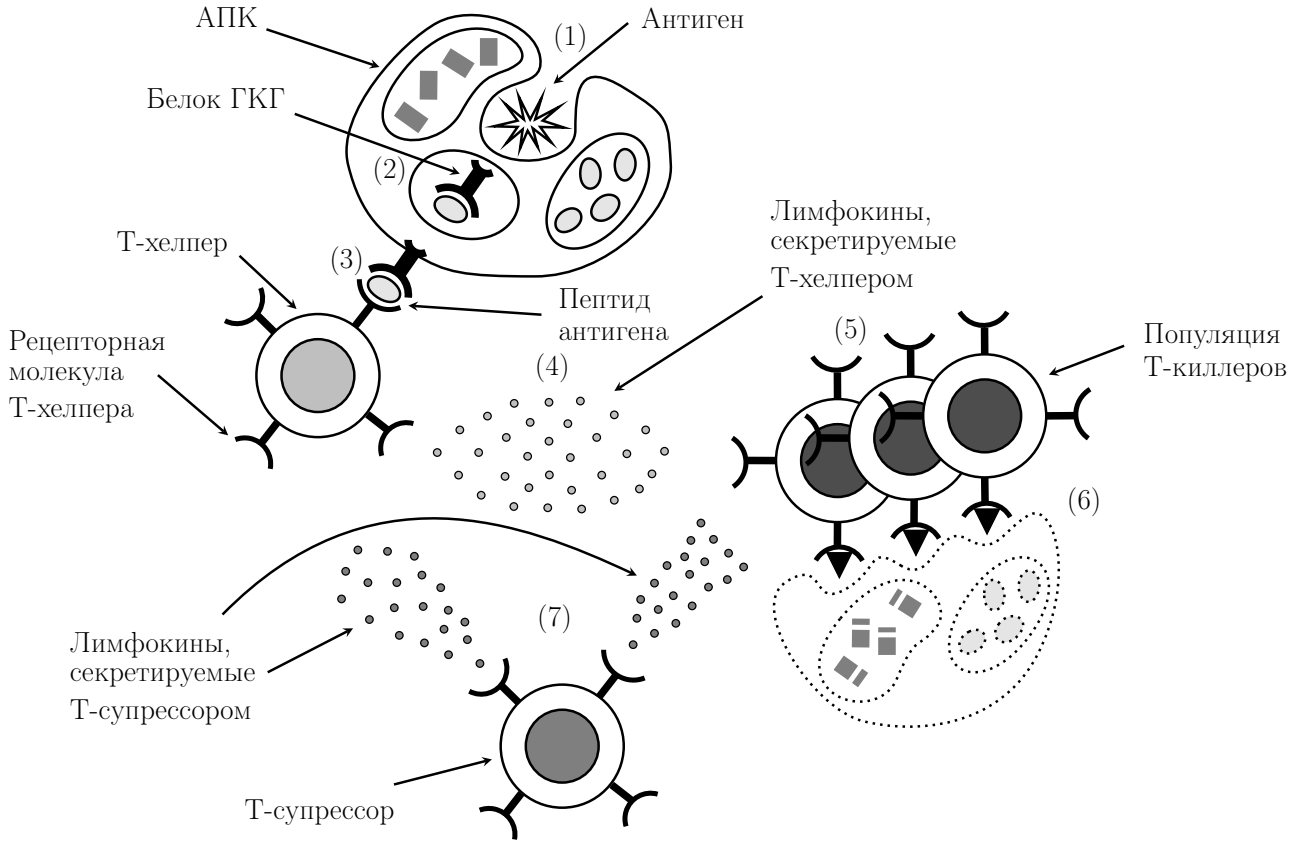


Рисунок 2.1 — Клеточный иммунный ответ

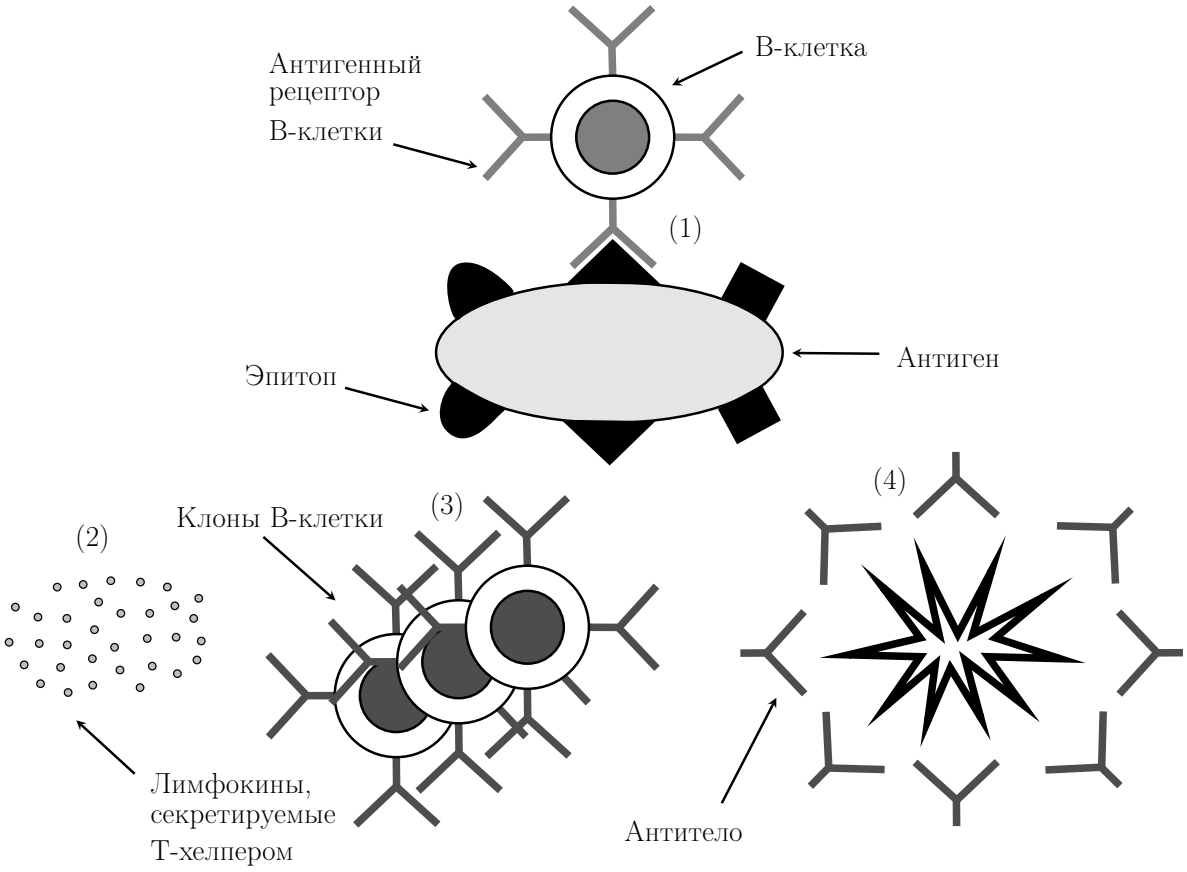


Рисунок 2.2 — Гуморальный иммунный ответ

Для имитации поведения компонентов и процессов их взаимодействия в естественной иммунной системе было предложено множество математических концепций и моделей, объединенных общей направленностью в виде решения задач интеллектуальной обработки данных. Направление вычислительного интеллекта, в котором исследуются вдохновленные идеями об иммунных положениях подходы, получило название искусственных, или вычислительных, иммунных систем, имеющих на данный момент широкую популярность во многих прикладных областях информатики.

Модель распознавания «свой-чужой» [104] базируется на процессе созревания Т-клеток в тимусе. Для того чтобы описать формирование и функционирование таких клеток (детекторов), применяется алгоритм отрицательного отбора, основанный на предположении о том, что априорно известна информация лишь о положительных объектах, которые рассматриваются как „свои“ (отличающиеся от них объекты именуются „чужими“). Данный алгоритм в классическом исполнении состоит из двух этапов: генерация детекторов и обнаружение „чужих“ объектов. Процесс создания детекторов включает последовательность следующих шагов, представленных в алгоритме 1.

Алгоритм 1: Генерация детекторов при помощи отрицательного отбора

Входные данные: Множество „своих“ объектов S

Выходные данные: Иммунные детекторы R

1 $R := \varnothing$

2 случайная генерация набора детекторов R'

3 исключение из набора R' тех детекторов, которые соответствуют какому-либо объекту из множества S

4 $R := R \bigcup R''$, где $R'' -$ отобранные на шаге 3 детекторы из R'

5 переход к шагу 2, если множество R содержит меньше предопределенного числа элементов; иначе останов

В результате образуется набор R, элементы которого покрывают пространство „чужих“ объектов. Второй этап заключается в сопоставлении всех детекторов, сохраненных в R, со входным объектом. Если входной объект соответствует какому-либо детектору из R, то данный объект распознается как „чужой“, что свидетельствует в большинстве случаев о наличии аномалии или изменении состояния/поведения наблюдаемого объекта.

Принцип клональной селекции моделирует поведение B-клеток в процессе иммунной реакции на антигенный стимул. Такое взаимодействие сопровождается выработкой клонов B-клетки, которые могут претерпевать мутации различной степени в зависимости от силы их связывания (меры аффинности) с каким-либо эпитопом антигена. Как и алгоритм отрицательного отбора, алгоритм клональной селекции относится к семейству популяционных алгоритмов, в которых особи, описывающие текущее решение задачи, подвергаются улучшениям и заменам и борются друг с другом за право быть отобранными в лучшие кандидаты. Первой реализацией алгоритма клональной селекции является CLONALG [69, 86, 87, 97], состоящий из следующих шагов, представленных в алгоритме 2.

Алгоритм 2: Клональная селекция

Входные данные: Множество антигенов A

Выходные данные: Популяция детекторов D

1 Подготовка входных и выходных данных: формирование множества антигенов A и генерация начальных детекторов D

2 Клональная селекция: отбор детекторов $D^* \subset D$ с наибольшим значением меры аффинности к множеству антигенов A

3 Клональная экспансия: клонирование (формирование точных копий) детекторов D^* с созданием популяций C, размер которых прямо пропорционален аффинности их предка (чем больше мера аффинности детектора, тем больший размер популяции производимых им клонов)

4 Созревание аффинности: формирование новых детекторов C' путем применения мутации к детекторам C с коэффициентом, обратно пропорциональным их аффинности (чем больше мера аффинности, тем меньше мутационные изменения), и отбор детекторов C^* с наибольшим значением меры аффинности из множества детекторов $C' \bigcup D^*$

5 Метадинамика: формирование множества детекторов D посредством замены детекторов C^* с наименьшими значениями меры аффинности новыми случайно сгенерированными

6 Переход к шагу 2 или останов в случае выполнения критерия оптимальности D

Авторы CLONALG [86, 87] подчеркивают следующие важные особенности алгоритма клональной селекции, отличающие его от генетического алгоритма: выполнение операторов мутации и отбора пропорционально аффинности особей, отсутствие оператора скрещивания и кодирование особей в виде двоичных строк вместо веществennозначного представления.

Одним из хорошо известных подходов, используемых для объяснения того, как взаимодействуют между собой B-клетки, является модель иммунных сетей, предложенная в 1974 г. физиологом Jerne [127]. Предполагалось, что динамика

развития иммунного ответа может зависеть не только от степени специфичности связывания между антигеном и рецептором В-клетки, но также и от концентрации других клеток определенного типа, циркулирующих в окрестности этого взаимодействия. Здесь под клетками определенного типа понимается набор лимфоцитов, имеющих одинаковое состояние дифференциации и производящих сходные антитела. Такая модель представляется как набор дифференциальных уравнений, описывающих скорость увеличения или уменьшения численности L_i лимфоцитов типа i [173]:

$$\left\{ \frac{dL_i}{dt} = \alpha - \beta L_i + L_i \sum_{j=1}^{\widetilde{m}} \varphi\left(E_j, K_j, t\right) - L_i \sum_{j=1}^{\widetilde{n}} \psi\left(I_j, K_j, t\right) \right\}_{i=1}^{\widetilde{k}},$$

где α задает скорость поступления новых лимфоцитов типа i, β соответствует скорости гибели лимфоцитов типа i. Функции φ и ψ выполняют роли возбуждающих и тормозящих сигналов соответственно и примимают в качестве первых аргументов наборы распознаваемых (E_j) и распознающих (I_j) лимфоцитов при связывании с рецепторами лимфоцитов типа i с силой K_j. Полное моделирование систем подобного вида существенно усложняется, если принимать во внимание, что в их биологических аналогах величины $\widetilde{k}$, $\widetilde{n}$ и $\widetilde{m}$ являются непостоянными и зависящими от параметра времени t. Поэтому для снижения накладных расходов при решении практических задач часто переходят к дискретным вариантам такой модели [84, 87]. Для упрощения процессов взаимодействия В-клеток Jerne были введены следующие обозначения. Паратоп — это участок антитела, который предназначен для распознавания эпитопа, находящегося на поверхности антигена, идиотоп — это область на поверхности антитела, которая подлежит распознаванию при помощи соответствующих ему паратопов, расположенных на других антителах. В итоге, взаимодействие лимфоцитов в иммунной системе может быть формализовано в виде сети переплетающихся связей между их антителами, паратопы которых распознают идиотопы, а идиотопы распознаются паратопами. После подобного распознавания соответствующее антитело может генерировать положительный сигнал, приводящий к активации

В-клеток, их пролиферации и продуцированию ими антител, либо отрицательный сигнал, приводящий к подавлению местных реакций и сохранению аутотолерантности в ответ на это распознавание.

Представленная в [173, 174] концепция пространства форм сужает „размах“ теории идиотипических сетей Jerne и предлагает рассматривать распознавание антигенов клетками иммунной системы исключительно на уровне их клеточных рецепторов. Величина аффинности при связывании между рецепторами антигена и лимфоцита или между рецепторами лимфоцитов зависит от того, насколько близко и в скольких позициях молекулы этих рецепторов могут контактировать друг с другом. Такой подход может быть интерпретирован как ассоциирование с каждым рецептором определенного замкового механизма и закрепление за ним набора взаимодополняющих его рецепторов, которые могут связываться с ним с той или иной степенью аффинности, причем чем выше значение аффинности между ними, тем больше вероятность того, что удалось подобрать подходящий ключ к данному замку. Набор функций, задающих свойства связывания этих молекул, называется обобщенной формой молекул, к ним Perelson и Oster относят следующие параметры: размеры выступов и впадин, их положение, заряд, наличие водородных связей между ними и прочих физико-химических реакций. Тем самым каждое антитело и эпитоп антигена могут быть представлены в виде точки $L = \langle l_1, l_2, \ldots, l_s \rangle$ в пространстве форм $P^s \subseteq \mathbb{R}^s$ [87], а окружающая точку L локальная область $V_{L,\varepsilon}$ будет содержать комплементарные ей обобщенные формы. В рамках данной концепции антитела способны распознавать не только один конкретный антиген, лежащий в пределах их радиуса досягаемости, но и другие схожие с ним антигены, располагающиеся внутри $V_{L,\varepsilon}$. Это свойство антител известно как кросс-реактивность. Параметр ε области $V_{L,\varepsilon}$ представляет собой порог аффинности рецептора со свойствами, которые задаются координатами точки L. Считается, что посредством правильного подбора параметра ε можно при помощи конечного числа точек, подобных L, охватить практически бесконечное число обобщенных антигенных форм.

Клеточный автомат моделирует поведение различных клеток иммунной системы в виде комбинаций ячеек внутри прямоугольной решетки ограниченных размеров [73]. Для дискретизации таких элементов используются непрерывные цепочки (последовательности) булевых значений. Так, каждой АПК сопоставлена в соответствие молекула ГКГ II класса, каждой В-клетке — молекула антигенного рецептора и молекула ГКГ II класса, каждой Т-клетке — ее рецепторная молекула и каждому антигену и антителу — молекулы эпитопа и пептида, и все из представленных молекул кодируются равноразмерными бинарными строками. Для описания хода развития иммунного ответа задаются правила, которые основаны на вычислении силы комплементарного взаимодействия между образами соответствующих молекул. Сила такого связывания прямо пропорционально зависит от количества ячеек, для битового содержимого которых оператор исключающего „или“ равен 1. Тем самым чем в больших позициях с одинаковыми индексами различаются сравниваемые представления молекул, тем больше вероятность возникновения между ними связывания.

Теория опасности [47, 153] применительно к процессу реагирования лимфоцитов на антигенные раздражители позволяет избавиться от некоторых противоречивых ситуаций, присущих модели распознавания „свой-чужой“. В частности, в рамках этой модели необъяснимыми являются следующие вопросы, возникающие при детальном изучении принципов функционирования естественных иммунных систем: (1) почему в желудке и кишечнике организма пища и чужеродные бактерии не вызывают иммунных реакций, (2) могут ли ранее дифференцированные и созревшие тимоциты корректно идентифицировать собственные клетки и ткани организма, состав которых динамически меняется, (3) чем обосновываются случаи атак собственных клеток организма детекторами иммунной системы (природа аутоиммунных заболеваний). В попытке решить эти вопросы Matzinger, главный идеолог теории опасности, вводит новые понятия, добавляющие ряд ограничений на классические взгляды иммунологии [169, 199], и утверждает, что иммунные механизмы защиты запускаются в ответ на возникновение специальных сигналов, источниками которых являются клетки,

подверженные стрессу или повреждениям. Предлагается рассматривать понятие „своих“ объектов как их надмножество в виде набора всех безвредных для организма элементов, возможно, включающих и антигенные микроорганизмы, но присутствие которых не вызывает некроза (незапланированной аномальной гибели) соседних тканей. Тем самым иммунная система вырабатывает толерантность к некоторым видам антигенов, например паразитических червей. Соответственно под „чужими“ объектами понимаются любые субстанции, в окрестности которых клетки индуцируют сигналы тревоги о поражении местного органа. Вокруг каждого такого сигнала формируется зона опасности, по прибытии в которую АПК захватывают антигены, перемещаются в лимфоузлы и предъявляют остатки антигенов Т-клеткам. Поведение лимфоцитов, находящихся в пределах этой зоны, контролируется дуальным механизмом выработки сигналов [66]: первый сигнал соответствует распознаванию антигена этим лимфоцитом, второй сигнал (костимуляция) закрепляет подтверждение о действительной чужеродности антигена. Более конкретно, первый сигнал индуцируется в результате связывания рецепторов лимфоцитов с частями антигена, второй сигнал генерируется хелперными Т-клетками в результате их взаимодействия с АПК, что приводит к стимуляции Т-киллеров и синтезу главных продуктов В-клеток, антител (иммуноглобулинов), препятствующих дальнейшей активности антигена. Такая двойная активация предостерегает организм от случайных срабатываний иммунных детекторов против „своих“ объектов.

Алгоритм дендритных клеток является одной из первых практических реализаций процесса, в котором моделируются и сигналы опасности, и их приемники. Основные его элементы, дендритные клетки, представляющие собой АПК, выполняют функции инструктирования иммунных реакций, начиная с самого раннего этапа, распознавания антигена, и заканчивая последним этапом, его разрушением, и служат в роли промежуточного звена между врожденной и адаптивной иммунными системами [112]. При патрулировании тканей (мест сбора данных) дендритной клеткой ее рецепторы являются особенно чувствительными к наличию окружающих ее сигналов: в зависимости от собственного

текущего состояния зрелости и концентрации перехватываемых ей сигналов она подвергается дифференциации, приводящей к смене ее состояния, возможной миграции в лимфатический узел (место для анализа данных) и активации или деактивации Т-клеток. Зрелая дендритная клетка, в отличие от полузрелых и незрелых аналогов, соответственно и вырабатывает сильнодействующий цитокин (интерлейкин-12), направленный именно на положительную стимуляцию Т-клеток, и обладает увеличенной по площади поверхностью, характеризующейся бóльшим числом выступов и способствующей представлению Т-клеткам большего числа захваченных антигенов. Выходные значения, соответствующие костимуляторным молекулам, полузрелым и зрелым сигналам, рассчитываются по формуле: $\frac{W_P \cdot C_P + W_S \cdot C_S + W_D \cdot C_D \cdot (1+IC)}{W_P + W_S + W_D}$, где коэффициенты W_P, W_S, W_D подбираются эмпирическим путем для каждого состояния зрелости дендритной клетки и составляют элементы матрицы сигналов, а значения C_P, C_S, C_D и IC представляют собой соответственно патогенные сигналы, генерируемые вредоносными бактериями и указывающие в большинстве случаев на наличие аномалии, безопасные сигналы, генерируемые в результате апоптоза клеток и свидетельствующие о здоровом окружении, сигналы опасности, генерируемые в результате некроза клеток и увеличивающие вероятность возникшей аномальной ситуации, и воспалительные цитокины, генерируемые в ответ на повреждение ткани и усиливающие эффекты предыдущих сигналов. Алгоритм дендритных клеток основан на следующих действиях [113], представленных в алгоритме 3.

Алгоритм 3: Алгоритм дендритных клеток

Входные данные: Коэффициенты W_P, W_S, W_D
Выходные данные: Дендритные клетки

1 создание начальной популяции дендритных клеток
2 выбор очередной дендритной клетки из популяции и инициализация параметров дендритной клетки
3 формирование подвыборки антигенов для дендритной клетки
4 обновление матрицы сигналов для дендритной клетки
5 вычисление выходных сигналов дендритной клетки при представлении собранных ей антигенов и аккумулирование этих выходных сигналов
6 сравнение костимуляторного выходного сигнала дендритной клетки с ее порогом миграции (удаление дендритной клетки в лимфатический узел и ее замена новой в случае превышения этим сигналом порога миграции)
7 завершение цикла генерации дендритных клеток в случае обработки всех антигенов или превышения числа заданных итераций, иначе переход к шагу 2
8 распознавание антигена через вычисление степени его аномальности как отношение накопленного зрелого сигнала к сумме накопленных полузрелых и зрелых сигналов

В отличие от данного алгоритма, алгоритм толл-подобных рецепторов дополнительно учитывает также и механизм взаимодействия дендритных клеток с Т-клетками [48, 162]. Условно процесс, описываемый этим алгоритмом, может быть разбит на три этапа, представленных в алгоритме 4.

Алгоритм 4: Алгоритм толл-подобных рецепторов

Входные данные: Антигены

Выходные данные: Т-клетки

1 Первый этап включает инициализацию сигнальных рецепторов незрелых дендритных клеток произвольными значениями сигналов опасности и инициализацию антигенных рецепторов наивных Т-клеток произвольными значениями аномальных антигенов, где оба обучающих набора данных получены в результате отрицательного отбора значений сигналов и антигенов

2 На втором этапе незрелые дендритные клетки, получившие хотя бы один сигнал опасности в ткани в течение заданного промежутка времени, превращаются в зрелые клетки и мигрируют в лимфатический узел, остальные клетки становятся полузрелыми

3 На третьем этапе захваченные зрелыми и полузрелыми дендритными клетками антигены представляются наивным Т-клеткам, причем активация последних происходит только в том случае, когда их антигенные рецепторы соответствуют данному антигену и сам антиген презентируется именно зрелой дендритной клеткой

Результатом работы данного алгоритма являются наборы Т-клеток с иммунокомпетентными рецепторами, случай связывания которых с антигеном, представленным зрелой дендритной клеткой, интерпретируется как наличие выявленной аномалии. Существенные технические различия алгоритма дендритных клеток и алгоритма толл-подобных рецепторов заключаются в отсутствии процесса настройки значений сигналов для первого алгоритма и использовании только двоичных сигналов для последнего алгоритма.

В [135] представлена многослойная иммунная модель, состоящая из слоя свободных антител, слоя В-клеток и слоя памяти. Выборочные элементы первого слоя выполняют процесс связывания со входным антигеном. На втором слое осуществляется клональная селекция В-клеток. Клетки памяти, размещенные на последнем слое, обеспечивают поддержание шаблонных форм, сгенерированных на предыдущих слоях. Преимуществом такой модели является уменьшение числа ложных срабатываний за счет многократного анализа структуры антигена и сохранения на последнем слое детекторов памяти, полученных в результате клональной селекции В-клеток.

В таблице 3 представлены общие сведения о рассмотренных моделях.

Таблица 3 — Модели и концепции искусственных иммунных систем

Иммунная модель/концепция	Основоположники и фундаментальные работы	Биологические прототипы	Моделируемые процессы
Модель „свой-чужой“	E. Metchnikoff (1883 г.) и P. Ehrlich (1908 г., Нобелевская премия); S. Forrest, A.S. Perelson, L. Allen, R. Cherukuri [104] (1994 г.)	Т-клетки, антигены	Дифференцировка Т-клеток в тимусе, клеточный апоптоз
Принцип клональной селекции	F.M. Burnet [71] (1959 г.) и P.B. Medawar (1960 г., Нобелевская премия); L.N. de Castro, F.J. Von Zuben [86] (2000 г.)	В-клетки, антигены	Созревание В-клеток в костном мозге, клональная пролиферация, соматическая гипермутация, создание клеток памяти
Иммунные сети	N.K. Jerne [127] (1974 г.), G.J.F. Köhler и C. Milstein (1984 г., Нобелевская премия)	В-клетки, антигены	Иммунологическая память, подавление и стимуляция связанных между собой клеток
Модель пространства форм	A.S. Perelson, G.F. Oster [174] (1979 г.)	В-клеточные рецепторы или антитела, эпитопы	Кросс-реактивность антител
Клеточные автоматы	J. Conway (1970 г.); M. Kaufman, J. Urbain, R. Thomas [131] (1985 г.); F. Celada, P.E. Seiden [73] (1992 г.)	Рецепторы лимфоцитов, молекулы ГКГ II класса, эпитопы, пептиды	Взаимодействие клеток на основе принципа комплементарности
Теория опасности	P. Matzinger [153] (1994 г.); P. Bretscher, M. Cohn [66] (1970 г.); U. Aickelin, S. Cayzer [47] (2002 г.)	Дендритные клетки, Т-клетки, антигены, цитокины	Клеточный некроз, дифференциация дендритных клеток, активация Т-клеток через сигнал костимуляции
Многослойная иммунная модель	T. Knight, J. Timmis [135] (2002 г.)	В-клетки, антитела, антигены, клетки памяти	Клональная селекция В-клеток

Рассмотрим некоторые из существующих исследовательских прототипов сетевых СОА, построенных на основе принципов искусственных иммунных систем. Отметим, что все из этих систем являются распределенными как наиболее универсальный и распространенный подход, используемый для построения СОА и организации ее составных компонентов в единую управляющую систему.

Система libtissue [113] предоставляет клиент-серверное окружение для разработки и тестирования иммунных алгоритмов, основанных на принципах теории опасности. Роль клиентов заключается в сборе сигналов и антигенов, а также их передаче на серверную сторону по протоколу SCTP. Сервер, в свою очередь, размещает популяции клеток вместе с образами объектов, полученных от клиентов, внутри специально выделенного блока — отделения ткани, в котором клетки, сигналы и антигены взаимодействуют между собой по заданному пользователем алгоритму. В [111] предложенная система используется для обнаружения SYN-сканирований, выполняемых без полного установления соединения. С этой целью было выделено семь сигналов, численно выраженных различными значениями среднего размера, скорости и интенсивности передачи сетевых пакетов специальных типов, а также признаком множественных удаленных входов в контролируемую систему под учетной записью суперпользователя.

Система LISYS (Lightweight Intrusion detection SYStem) [121] представляет собой СОА, которая ориентирована на работу исключительно в широковещательных компьютерных сетях. Для выполнения задач распределенного мониторинга сетевого окружения она использует в качестве основы иммунные детекторы, представленные в виде двоичных строк, и правила r-непрерывных битов. Согласно этому правилу, если структуры обученного детектора и тестового объекта совпадают по меньшей мере в r последовательно идущих друг за другом позициях, то считается, что объект представляет собой аномалию (антиген). Каждый детектор кодируется в виде кортежа, содержащего IP-адреса источника и назначения, а также службу TCP-соединения. Набор данных для обучения представлен аналогичным способом, где каждый элемент соответствует соединению, чьи представленные характеристики наблюдаются с высокой частотой в локальной сети. И наоборот, если тройка задает редкое по появлению соединение, то она относится к набору „чужих“ объектов.

Архитектура сетевой системы обнаружения атак, представленной в [133] является двухуровневой и включает в свой состав первичную СОА и несколько вторичных СОА. Каждая из этих СОА снабжается коммуникатором для возмож-

ности обмена данными по сети между друг другом. Первичная СОА рассматривается как костный мозг или тимус, генерирующий зрелые иммунные детекторы. Для создания таких детекторов внутри этой СОА заложены процессы эволюции библиотеки генов и отрицательного отбора. Роль первого процесса заключается в создании и пополнении специального хранилища атрибутов детекторов, генерирующих события об обнаружении сетевых аномалий, — библиотеки генов. Добавление новой записи в библиотеку генов при достижении ее максимальных размеров зависит от степени пригодности соответствующего детектора, что позволяет искусственной иммунной системе динамически обучаться на актуальных данных, собранных в различные моменты времени. Роль второго процесса заключается в формировании набора детекторов, предварительно сгенерированных на основе данных библиотеки генов и операторов мутации и впоследствии отфильтрованных по алгоритму отрицательного отбора, и передаче его во вторичные СОА. Во вторичных СОА выполняется процесс клональной селекции тех детекторов, которые обнаружили аномальную активность в сетевом трафике. Одна из копий клонированных детекторов сохраняется в качестве детектора памяти на текущей СОА, а остальные распространяются на другие СОА.

Предлагаемая в [83] СОА представляет собой мультиагентную систему, которая обеспечивает выявление аномальной активности на различных уровнях: уровне пользователя, системном уровне, уровне процессов и пакетном уровне. Взаимодействующие между собой и с окружающей средой мобильные компоненты (иммунные агенты) выполняют строго закрепленные за ними роли на этих уровнях и координируют свои действия для выработки общего решения. Деятельность самих агентов СОА имеет прямую аналогию с биологическими процессами в иммунной системе и организуется в иерархическом стиле. В системе выделяется несколько режимов ее функционирования. В режиме сканирования СОА отслеживает возникновение различных событий в сетевом окружении. В режиме распознавания СОА регистрирует запросы отдельных агентов, сигнализирующих о наличии аномалии. В режиме выработки решения СОА активирует специальных агентов, устраняющих системную или сетевую угрозу.

2.2 Модель искусственной иммунной системы на базе эволюционного подхода

Разработанная модель искусственной иммунной системы на базе эволюционного подхода $AISEA$ (Artificial Immune System based on Evolutionary Approach) представляется следующим образом:

$$AISEA = \langle \mathcal{D}_\mathcal{T}, \mathcal{D}_\mathcal{M}, \mathcal{S}_\mathcal{A}, \mathcal{S}_\mathcal{N}, \mathcal{G}, R, \Psi \rangle,$$

где $\mathcal{D}_\mathcal{T} \subset \mathcal{D}$ — набор временных иммунных детекторов, $\mathcal{D}_\mathcal{M} \subset \mathcal{D}$ — набор иммунных детекторов памяти, $\mathcal{S}_\mathcal{A} \subset \mathcal{S}$ — обучающий набор, состоящий из аномальных экземпляров („чужих“ объектов), $\mathcal{S}_\mathcal{N} \subset \mathcal{S}$ — тестирующее множество, состоящее из нормальных экземпляров („своих“ объектов), $\mathcal{D} = \mathcal{D}_\mathcal{T} \bigcup \mathcal{D}_\mathcal{M}$ — набор иммунных детекторов, $\mathcal{G} = \{G_1, \ldots, G_K\}$ — стратегии генетической оптимизации иммунных детекторов, $R : \mathcal{D} \times 2^{\mathcal{S}_\mathcal{A}} \times 2^{\mathcal{S}_\mathcal{N}} \times \mathcal{G} \to \mathcal{D}$ — правило обучения иммунных детекторов, $\mathcal{S}$ — набор всевозможных входных объектов, $\Psi : \mathcal{D} \times \mathcal{S} \to \mathbb{R}_+$ — функция вычисления аффинности (правило соответствия) между иммунным детектором $d \in \mathcal{D}$ и тестовым объектом $s \in \mathcal{S}$, где $\mathbb{R}_+ = \mathbb{R} \bigcap [0, +\infty)$.

Каждый иммунный детектор $d \in \mathcal{D}$ представляется как кортеж следующего вида:

$$d = \langle representation, threshold, life\_time, state \rangle,$$

где $representation \in \{BitString, RealVector, NeuralNetwork, PetriNet, \ldots\}$ — внутреннее представление (внутренняя структура) иммунного детектора d, который может быть задан как двоичная строка с правилом r-непрерывных битов ($BitString$), вещественнозначный вектор ($RealVector$), нейронная сеть ($NeuralNetwork$), сеть Петри ($PetriNet$) и т.д., $threshold \in \mathbb{R}_+$ — порог активации иммунного детектора d, $life\_time \in \mathbb{R}_+^*$ — срок жизни иммунного детектора d, $state \in \{immature, semimature, mature, memory\}$ — текущее

состояние иммунного детектора d, которое может представлять собой незрелое, полузрелое, зрелое состояние или состояние, соответствующее детектору памяти. Здесь $\mathbb{R}_+^* = \mathbb{R}_+ \bigcup \{+\infty\}$.

Общий подготовительный процесс построения параметров искусственной иммунной системы $AISEA$ может быть описан следующим образом:

1. Выбор внутренней структуры для каждого детектора $d \in \mathcal{D}$: $representation$.
2. Формирование обучающего набора данных $\mathcal{S}_{\mathcal{A}}$, содержащего заранее отобранные „чужие“ объекты.
3. Формирование тестирующего набора данных $\mathcal{S}_{\mathcal{N}}$, содержащего заранее отобранные „свои“ объекты.
4. Выбор стратегии генетической оптимизации иммунных детекторов.
5. Выбор алгоритма обучения R иммунных детекторов $\mathcal{D}$ в зависимости от их внутреннего представления.
6. Выбор правила соответствия Ψ между иммунным детектором и входным объектом.

Разработанная модель искусственной иммунной системы — это набор иммунных детекторов, представленных в виде временных детекторов и детекторов памяти, в совокупности с заданным алгоритмом их обучения, который принимает в качестве входных аргументов настраиваемый детектор d, подмножества двух наборов данных (набора $\mathcal{S}_{\mathcal{A}}$, состоящего из „чужих“ объектов, и набора $\mathcal{S}_{\mathcal{N}}$, состоящего из „своих“ объектов), а также стратегию генетической оптимизации. Причем набор данных $\mathcal{S}_{\mathcal{A}}$ предназначается для первой предварительной настройки иммунных детекторов, а роль набора данных $\mathcal{S}_{\mathcal{N}}$ заключается в фильтрации обученных детекторов. Стратегии генетической оптимизации иммунных детекторов включают некоторый набор генетических операторов (кроссовера, мутации, инверсии) и их комбинаций для изменения параметров иммунного детектора после его клонирования. Правило обучения иммунных детекторов представляет собой двухшаговую процедуру. На первой фазе иммунные детекторы обучаются исключительно на элементах набора данных $\mathcal{S}_{\mathcal{A}}$ и подвергаются кло-

нальной селекции, в процессе которой созданные копии иммунных детекторов мутируют согласно выбранной стратегии $G' \in \mathcal{G}$. Из набора детекторов, представленного произведенным потомством и исходным детектором, отбираются только те детекторы, которые являются наиболее пригодными по отношению к элементам набора $\mathcal{S}_{\mathcal{A}}$ согласно выбранному правилу соответствия Ψ. Данная фаза повторяется несколько раз для формирования полузрелых детекторов. На второй фазе обучения эти детекторы проверяются на соответствие „своим“ объектам: те из них, которые ложно активируются, уничтожаются, заново инициализируются и обучаются. В рамках данной модели каждый иммунный детектор подвергается нескольким этапам дифференцировки. На рисунке 2.3 представлены процессы жизненного цикла иммунного детектора вместе с указанными этапами его дифференцировки.

Рисунок 2.3 — Процессы жизненного цикла иммунного детектора

В начале своего развития каждый детектор инициализируется произвольным образом в соответствии со своим внутренним представлением. В процессе функционирования СОА временные детекторы осуществляют запись парамет-

ров атак в обновляемый набор аномальных данных $\mathcal{S}_{\mathcal{A}^*}$ в случае их обнаружения. Все иммунные детекторы, кроме детекторов памяти, наделены конечным сроком жизни. Если в течение данного им срока жизни они не выявили ни одной атаки, они подвергаются повторному обучению на расширенном наборе данных, содержащем элементы $\mathcal{S}_{\mathcal{A}}$ и $\mathcal{S}_{\mathcal{A}^*}$. В то же время если иммунный детектор распознал соединение как аномальное, его срок жизни увеличивается. В отличие от них, детекторы памяти обладают бесконечным сроком жизни и не принимают участия в наполнении обновляемого набора аномальных данных.

Для обнаружения каждого класса атаки $C \in \mathcal{C}$ выделяется несколько иммунных детекторов, объединяющихся в класс детекторов $\mathcal{D}_{\zeta(C)}$ ($\bigcup\limits_{C\in\mathcal{C}} \mathcal{D}_{\zeta(C)} = \mathcal{D}$). Каждый из детекторов $d \in \mathcal{D}_{\zeta(C)}$ использует бутстреп-обучение на разных случайных подвыборках множеств $\mathcal{S}_{\mathcal{A}}$ и $\mathcal{S}_{\mathcal{N}} - \mathcal{S}_{\mathcal{A}}^{(d)}$ и $\mathcal{S}_{\mathcal{N}}^{(d)}$, которые, возможно, содержат некоторые дублирующиеся и переупорядоченные объекты из исходных наборов. Тем самым достигается разнообразие иммунных детекторов внутри $\mathcal{D}_{\zeta(C)}$. Соответственно под q-ой группой детекторов понимается набор детекторов $\mathcal{D}^{(q)}$ ($\bigcup\limits_{q=1}^{m} \mathcal{D}^{(q)} = \mathcal{D}$, m — число классов атак), полностью покрывающих заданное множество классов атак (рис. 2.4). Если детектор d реагирует на какой-либо элемент $s' \in \mathcal{S}_{\mathcal{N}}^{(d)}$, т.е. если $\exists s' \in \mathcal{S}_{\mathcal{N}}^{(d)}\ \Psi(d,s') > \min\limits_{s\in\mathcal{S}_{\mathcal{A}}^{(d)}} \Psi(d,s)$, то детектор d подвергается апоптозу и замене новым произвольно сгенерированным детектором. Для каждого класса атаки $C \in \mathcal{C}$ определяется ровно один детектор памяти $d_m^{(C)}$ — детектор, который удовлетворяет требованию наибольшей степени приспособленности к распознаванию объектов $\mathcal{S}_{\mathcal{A}}^{\left(d_m^{(C)}\right)} \bigcap C$. Тем самым набор иммунных детекторов памяти может быть определен следующим образом:

$$\mathcal{D}_{\mathcal{M}} = \bigcup_{C\in\mathcal{C}} \left\{ \operatorname{Arg}\max_{d\in\mathcal{D}_{\zeta(C)}} \left(\frac{\sum\limits_{s\in\mathcal{S}_{\mathcal{A}}^{(d)}\bigcap C} \Psi(d,s)}{\#\left(\mathcal{S}_{\mathcal{A}}^{(d)} \bigcap C\right)} \right) \right\}.$$

Набор временных иммунных детекторов задается следующим образом:

$$\mathcal{D}_{\mathcal{T}} = \mathcal{D} \setminus \mathcal{D}_{\mathcal{M}}.$$

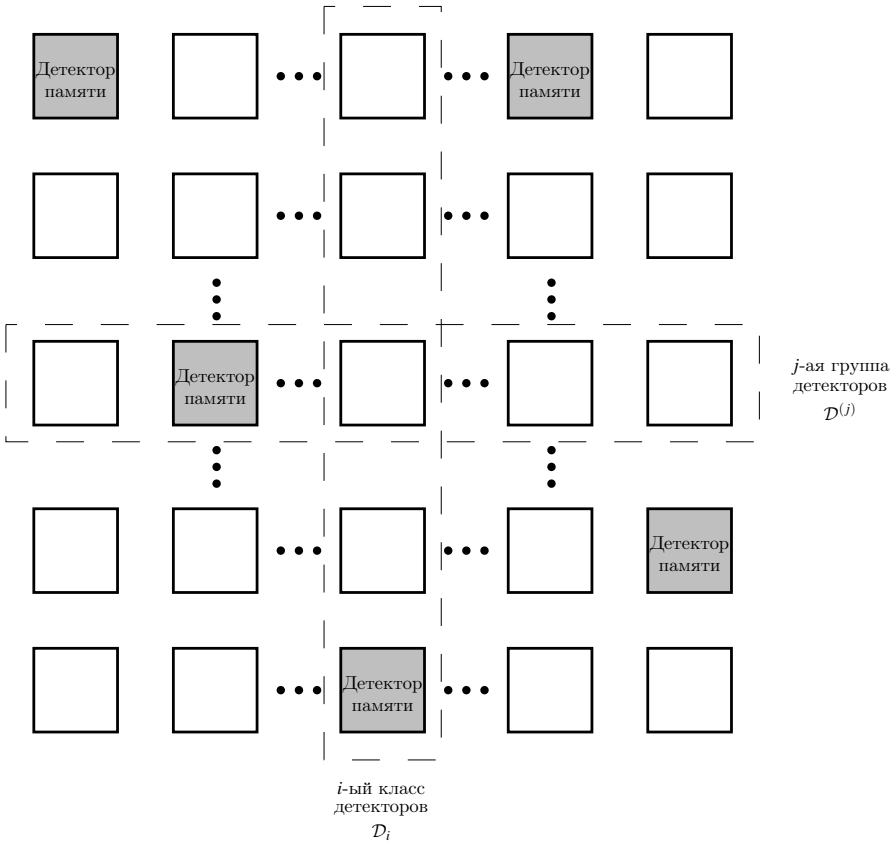


Рисунок 2.4 — Классы и группы иммунных детекторов

Порог активации иммунного детектора $d \in \mathcal{D}$, обученного на наборах $\mathcal{S}_{\mathcal{A}}^{(d)}$ и $\mathcal{S}_{\mathcal{N}}^{(d)}$, вычисляется следующим образом:

$$threshold = \frac{\overbrace{\min\limits_{s \in \mathcal{S}_{\mathcal{A}}^{(d)}} \Psi\left(d,s\right)}^{h_d^-} + \overbrace{\max\limits_{s \in \mathcal{S}_{\mathcal{N}}^{(d)}} \Psi\left(d,s\right)}^{h_d^+}}{2}.$$

Данная формула применяется для вычисления порога активации только тех детекторов d, которые после обучения на множестве аномальных данных $\mathcal{S}_{\mathcal{A}}^{(d)}$ не имеют ложных срабатываний на множестве нормальных данных $\mathcal{S}_{\mathcal{N}}^{(d)}$, т.е. $\forall s' \in \mathcal{S}_{\mathcal{N}}^{(d)}\ \Psi(d,s') < h_d^-$. Если такое условие выполняется, то появляется дополнительный зазор, равный величине $h_d = h_d^- - h_d^+ > 0$, и тем самым возникает возможность „сдвинуть“ граничное значение h_d^-, обеспечивающее реагирование детектора d на любой „чужой“ объект $s \in \mathcal{S}_{\mathcal{A}}^{(d)}$, на величину $\frac{h_d^- - h_d^+}{2}$ в сторону h_d^+: $threshold = h_d^- - \frac{h_d^- - h_d^+}{2} = \frac{h_d^- + h_d^+}{2}$.

Классификация объекта $s \in \mathcal{S}$ при помощи рассмотренной модели $AISEA$ представлен в алгоритме 5.

Алгоритм 5: Классификация сетевых соединений при помощи $AISEA$

1. Вычисление для каждого иммунного детектора $d \in \mathcal{D}$ значения его активации $a_d = \Psi(d,s) - threshold$. Считается, что, если $a_d \geqslant 0$, то детектор d является активировавшимся, в противном случае соответствующий детектор не реагирует на входной объект s.
2. Мажоритарное голосование внутри каждого класса детекторов $\mathcal{D}_{\zeta(C)}$. Если $\underbrace{\sum_{d \in \mathcal{D}_{\zeta(C)}} [a_d \geqslant 0]}_{A_C} > \underbrace{\sum_{d \in \mathcal{D}_{\zeta(C)}} [a_d < 0]}_{B_C = \#\mathcal{D}_{\zeta(C)} - A_C}$, то s распознается как „чужой“ объект. Если $A_C < B_C$, то s распознается как „свой“ объект. В случае наличия конфликтов, т.е. $A_C = B_C$, s классифицируется как „чужой“ объект, если $a_{d_m^{(C)}} \geqslant 0$, и s классифицируется как „свой“ объект, если $a_{d_m^{(C)}} < 0$, где $d_m^{(C)} \in \mathcal{D}_{\mathcal{M}} \bigcap \mathcal{D}_{\zeta(C)}$, т.е. $d_m^{(C)}$ — детектор памяти, обученный для распознавания „своего“ объекта и „чужого“ объекта из класса C.
3. Формирование множества классов активировавшихся иммунных детекторов $\left\{\mathcal{D}_{\zeta(C')}\right\}_{C' \in \mathcal{C}^*}$, которые распознают входной объект s как „чужой“ объект, где $\mathcal{C}^* = \left\{ C' \,\middle|\, C' \in \mathcal{C} \wedge \left(A_{C'} > B_{C'} \vee \left(A_{C'} = B_{C'} \wedge a_{d_m^{(C')}} \geqslant 0 \right) \right) \right\} \subset \mathcal{C}$.
4. Определение класса объекта s. Если $\mathcal{C}^* = \varnothing$, то объект s относится к классу „своих“ объектов. Если $E_{\mathcal{C}^*} \leftrightharpoons \max\limits_{C' \in \mathcal{C}^*} A_{C'}$ достигается в одной единственной точке, то класс объекта s — это $\arg E_{\mathcal{C}^*}$, иначе класс объекта s — это $\arg \max\limits_{C' \in \{\arg E_{\mathcal{C}^*}\}} \sum\limits_{d \in \mathcal{D}_{\zeta(C')}} a_d \cdot [a_d \geqslant 0]$.

Данный алгоритм основан на сравнении величины аффинности иммунных детекторов с их индивидуально настроенными порогами активации и учете одинаковых голосов, полученных от большей части детекторов. В случае возникновения конфликтов при различении между нормальным и аномальным классом (шаг 2) решающий голос отдается детектору памяти. Если же после этого сохраняется конфликт на уровне группы детекторов, то принимается во внимание сумма величин аффинности именно активировавшихся в ответ на данный стимул (входной объект) иммунных детекторов (шаг 4). Входной объект является „своим“ тогда и только тогда, когда $\forall C \in \mathcal{C}\ A_C < B_C \vee \left(A_C = B_C \wedge a_{d_m^{(C)}} < 0 \right)$.

В качестве основы для данной модели использовались модель с жизненным циклом (M_1), предложенная Hofmeyr и Forrest [121, 122], и модель с библиотекой генов (M_2), предложенная Kim и Bentley [133]. Разработанная модель $AISEA$ (M_3) была дополнена рядом усовершенствований, а именно двухступенчатым алгоритмом обучения иммунных детекторов, механизмом автоматического подбора их порога активации, а также процедурой разрешения конфликтных случаев классификации одного объекта при помощи иммунных детекторов памяти. Сравнение этих трех моделей приведено в таблице 4. Знаком „+“ отмечены те характеристики, которые присущи соответствующей модели, знак „−“ означает отсутствие поддержки этой особенности у модели.

Таблица 4 — Сравнение иммунных моделей для обнаружения сетевых атак

Иммунная модель	Сравниваемые характеристики											
	Независимость от внутренней структуры иммунного детектора	Наличие клональной селекции и генетической оптимизации	Наличие отрицательного отбора	Автоматический подбор порога активации иммунного детектора	Наличие двухступенчатого алгоритма обучения иммунных детекторов	Наличие детекторов памяти	Наличие жизненного цикла лимфоцитов	Динамическое переобучение	Обучение детекторов на новых данных в процессе функционирования системы	Распределенность иммунных детекторов (передача их на другие сетевые узлы)	Поддержка мультиклассового обнаружения сетевых атак	Автономность иммунной системы (функционирование без привлечения оператора)
M_1	−	−	+	−	−	+	+	+	+	−	−	−
M_2	−	+	+	−	−	+	−	+	+	+	−	+
M_3	+	+	+	+	+	+	+	+	+	−	+	+

Разработанная модель M_3 является универсальной по отношению к внутреннему представлению иммунных детекторов, в то время как модель M_1 ориентирована на использование битовых строк в качестве иммунных детекторов, а модель M_2 — на кластерную дискретизацию полей (генов) иммунных детекторов. В модели M_1 отсутствует возможность клональной селекции детекторов, а также не предусмотрены операторы генетической мутации. Во всех из трех представленных моделей в контексте обнаружения сетевых атак использу-

ется динамически обновляемая популяция иммунных детекторов, что позволяет СОА адаптироваться к изменяющимся сетевым условиям в режиме ее функционирования. Однако здесь для модели M_1 вводится дополнительное ограничение: после активации детектора в результате накопления достаточного числа совпадений с антигенами должен быть сгенерирован внешний по отношению к системе сигнал со строны администратора (сигнал костимуляции); такой сигнал позволит активированному детектору получить шанс для вступления в пул детекторов памяти и возможного дальнейшего непрерывного анализа сетевого трафика. Для модели M_2 характерен распределенный механизм генерации детекторов: детекторы, выгодные с точки зрения распознавания аномалий, могут быть размножены на другие сетевые узлы для повышения общей производительности системы; это свойство отсутствует у моделей M_1 и M_3.

В работах [121] и [133] речь шла именно о применении алгоритма „свой-чужой“ при детектировании сетевых атак без последующей их классификации. Тем самым модели M_1 и M_2 ориентированы только на обучение бинарной классификации объектов, что с некоторой стороны может накладывать ограничения на применение этих моделей. Разработанная модель M_3 лишена этого недостатка за счет введения вышеописанного алгоритма классификации объектов.

Основными и новыми отличительными особенностями разработанной модели являются наличие двухступенчатого алгоритма обучения иммунных детекторов и автоматическое вычисление порога активации иммунных детекторов. Первая фаза обучения включает настройку иммунных детекторов, которая зависит от их внутреннего представления. Пример одного из алгоритмов такого обучения вместе с возможными для него стратегиями генетической оптимизации представлен в разделе 2.3 для случая $representation = NeuralNetwork$. Вторая фаза обучения — отбор иммунных детекторов, не реагирующих на „свои“ объекты. За счет использования такого приема в обучении, с одной стороны, отпадает необходимость ручного подбора порога активации, а, с другой стороны, возникает необходимость иметь в наличии набор „чужих“ объектов для обучения модели.

2.3 Алгоритм генетико-конкурентного обучения сети Кохонена

В данном исследовании в качестве внутреннего представления иммунных детекторов были выбраны сети Кохонена (самоорганизующиеся карты) [16] ($representation = NeuralNetwork$). Такие сети представляют собой двухуровневые структуры, в которых входные сигналы, распределенные на внешний слой сети, сравниваются с весами нейронов выходного слоя. Как правило, нейроны выходного слоя организуются таким образом, чтобы они формировали прямоугольную решетку размером $I \times J$ (рисунок 2.5). Для сетей Кохонена характерны обучение без учителя и классификация на основе соотнесения объектов к определенным областям на выходной решетке. После обучения сети в случае близкого соответствия входного вектора и весового вектора нейрона на выходной решетке первый вектор попадает в кластер, который образован этим нейроном и его окружением. Для интеграции сетей Кохонена в иммунную модель $AISEA$ был разработан модифицированный алгоритм конкурентного обучения сети Кохонена, дополненный введением генетических операторов для скрещивания и мутации весовых коэффициентов отдельных нейронов на выходном слое сети.

Обозначим через $\boldsymbol{w_{ij}}$ весовой вектор нейрона, который имеет координаты (i,j) на выходной решетке сети Кохонена (i — номер строки, j — номер столбца). Процесс обучения направлен на минимизацию половинной суммы квадратов расстояний между входными векторами $\{\boldsymbol{x_k}\}_{k=1}^{M}$ обучающей выборки и векторами $\boldsymbol{w_{ij}}$ нейронов выходной решетки ($1 \leqslant i \leqslant I$, $1 \leqslant j \leqslant J$).

$$E(\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}) = \frac{1}{2} \cdot \sum_{k=1}^{M} \left(D\left(\boldsymbol{w_{i_k^* j_k^*}},\boldsymbol{x_k}\right)\right)^2 \to \min_{\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}},$$

где $D(\boldsymbol{w},\boldsymbol{x}) = \sqrt{(\boldsymbol{w}-\boldsymbol{x})^T \cdot (\boldsymbol{w}-\boldsymbol{x})}$ — функция расстояния между парой векторов в евклидовом пространстве, $(i_k^*,j_k^*) = \overbrace{\arg\min_{\substack{1\leqslant i\leqslant I\\ 1\leqslant j\leqslant J}} D\left(\boldsymbol{w_{ij}},\boldsymbol{x_k}\right)}^{F_{IJ}(\boldsymbol{x_k}):\mathbb{R}^n\to\{1,\ldots,I\}\times\{1,\ldots,J\}}$ — координаты нейрона на выходном слое сети, веса которого наиболее близки к вектору $\boldsymbol{x_k}$.

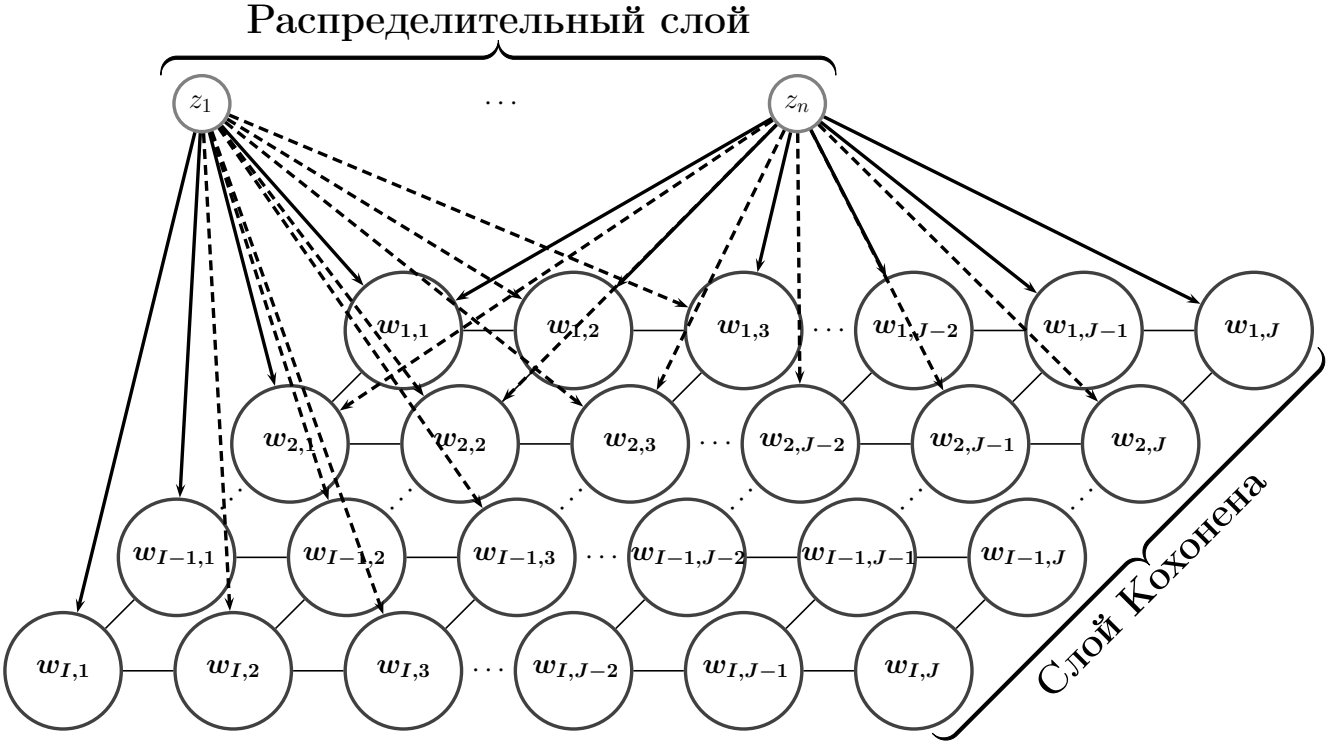


Рисунок 2.5 — Сеть Кохонена

Величина $\frac{1}{M} \cdot E(\boldsymbol{w_{11}}, \ldots, \boldsymbol{w_{IJ}})$ известна как погрешность векторного квантования [24]. Используя метод градиентного спуска, получаем следующую формулу для обновления весовых векторов $\boldsymbol{w_{ij}}$ ($1 \leqslant i \leqslant I$, $1 \leqslant j \leqslant J$):

$$\boldsymbol{w_{ij}}(t+1) = \boldsymbol{w_{ij}}(t) + \Delta \boldsymbol{w_{ij}},$$

$$\Delta \boldsymbol{w_{ij}} = -\kappa \cdot \frac{\partial E(\boldsymbol{w_{11}}, \ldots, \boldsymbol{w_{IJ}})}{\partial \boldsymbol{w_{ij}}} = -\kappa \cdot \sum_{k=1}^{M} \left([(i,j) = F_{IJ}(\boldsymbol{x_k})] \cdot (\boldsymbol{w_{ij}} - \boldsymbol{x_k}) \right) =$$

$$= \kappa \cdot \sum_{k=1}^{M} \left([(i,j) = F_{IJ}(\boldsymbol{x_k})] \cdot (\boldsymbol{x_k} - \boldsymbol{w_{ij}}) \right),$$

где κ — некоторая положительная константа или функция с областью значений $(0,1]$, задающая скорость обучения. Отметим, что в представленной формуле для обновления каждого конкретного весового вектора $\boldsymbol{w_{ij}}$ задействуется только часть векторов из обучающей выборки с наименьшей невязкой между каждым из них и $\boldsymbol{w_{ij}}$. Иными словами, вектор $\boldsymbol{w_{ij}}$ модифицируется тогда и только тогда, когда он является наиболее близким к обучающему вектору $\boldsymbol{x_k}$ в рамках заданного метрического пространства. Причем поправка вектора $\boldsymbol{w_{ij}}$ осуществляется

на величину, прямо пропорциональную разности между входным вектором $\boldsymbol{x_k}$ и весовым вектором $\boldsymbol{w_{ij}}$. Тем самым между нейронами на выходной решетке создается конкуренция за право быть отобранными в ближайшие по отношению ко входному вектору $\boldsymbol{x_k}$ кандидаты; нейрон, удовлетворяющий данному требованию, называется нейроном-победителем с координатами (i_k^*,j_k^*).

Отметим, что в случае нормированных векторов $\boldsymbol{w_{ij}}$ и $\boldsymbol{x_k}$ минимизация $E(\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}})$ равносильна максимизации суммы их скалярных произведений:

$$E(\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}) = \frac{1}{2}\cdot\sum_{k=1}^{M}\left(\boldsymbol{w}_{i_k^*j_k^*}-\boldsymbol{x_k}\right)^T\cdot\left(\boldsymbol{w}_{i_k^*j_k^*}-\boldsymbol{x_k}\right) =$$

$$= \frac{1}{2}\cdot\sum_{k=1}^{M}\big(\underbrace{\boldsymbol{w}_{i_k^*j_k^*}^T\cdot\boldsymbol{w}_{i_k^*j_k^*}}_{1}+\underbrace{\boldsymbol{x_k}^T\cdot\boldsymbol{x_k}}_{1}-2\boldsymbol{w}_{i_k^*j_k^*}^T\cdot\boldsymbol{x_k}\big) = M-\sum_{k=1}^{M}\boldsymbol{w}_{i_k^*j_k^*}^T\cdot\boldsymbol{x_k}\Rightarrow$$

$$\Rightarrow \min_{\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}} E(\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}) = \max_{\boldsymbol{w_{11}},\ldots,\boldsymbol{w_{IJ}}}\sum_{k=1}^{M}\boldsymbol{w}_{i_k^*j_k^*}^T\cdot\boldsymbol{x_k}-M.$$

Для смягчения конкуренции между нейронами вводится правило, позволяющее обновлять не только веса нейрона-победителя, но и других нейронов, лежащих в его окрестности. С этой целью ранее введенная характеристическая функция $[(i,j)=F_{IJ}(\boldsymbol{x_k})]$ заменяется экспоненциальной функцией Гаусса $\varphi(i,j)=\exp\left\{-\frac{(i-i_k^*)^2+(j-j_k^*)^2}{2\cdot\sigma^2}\right\}$, значение которой отражает затухающую зависимость изменения нейронных весов с увеличением расстояния от нейронов до нейрона-победителя на уровне их координат на выходной решетке. Чем ближе нейрон располагается к нейрону-победителю, тем с большим мультипликативным коэффициентом обновляются его веса. Параметр σ называется эффективной шириной окрестности [39] нейрона-победителя, который может быть интерпретирован как текущее значение радиуса окружения нейрона-победителя.

Особенностью алгоритма обучения сети Кохонена является уменьшение значения σ с течением времени: $\sigma(t)=\sigma_0\cdot\exp\left\{-\tau\cdot\frac{t}{T-1}\right\}$ $(t=0,\ldots,T-1)$. Здесь параметр σ_0 задает начальное значение радиуса окрестности нейрона-победителя, которое, как правило, устанавливается в $\sqrt{I^2+J^2}$, а параметр τ подбирается таким образом, чтобы на последней эпохе обучения обновлению

подвергалось как можно меньшее число весовых векторов нейронов или вовсе только один вектор нейрона-победителя. Тем самым $\tau = \ln(\sigma_0)$. Коэффициент скорости обучения κ выбирается таким образом, чтобы на начальных эпохах алгоритма весовые векторы большинства нейронов обновлялись с наибольшим темпом, а далее по мере увеличения числа эпох и сужения ширины окрестности осуществлялась модификация все меньшего числа векторов нейронов с более низкой скоростью. Использование такого приема позволяет строить кластеры, чьи элементы сначала приспосабливаются под общие характеристики аппроксимируемого множества, а затем уточняют его отдельные особенности. Наиболее распространенными представителями с такой характерно убывающей зависимостью являются функции $\kappa(t) = \kappa_0 \cdot (t+1)^{-1}$ [8] и $\kappa(t) = \kappa_0 \cdot \exp\{-\eta \cdot t\}$ [39].

Алгоритм конкурентного обучения сети Кохонена, дополненный введением генетических операторов, представлен в алгоритме 6.

Алгоритм 6: Алгоритм генетико-конкурентного обучения сети Кохонена

1. Задание параметров сети Кохонена (размера выходной решетки $I \times J$, числа эпох обучения $T \geqslant 1$, начальной ширины окрестности нейронов σ_0, коэффициентов τ, κ_0, η), зануление счетчика текущих итераций $t := 0$, инициализация весовых коэффициентов $\boldsymbol{w_{ij}}$ ($1 \leqslant i \leqslant I$, $1 \leqslant j \leqslant J$) нейронов выходной решетки случайным образом, подготовка обучающих данных $\{\boldsymbol{x_k}\}_{k=1}^{M}$, выбор стратегии оптимизации G' весовых коэффициентов нейронов выходной решетки.
2. Вычисление текущей ширины окрестности: $\sigma(t) = \sigma_0 \cdot \exp\left\{-\tau \cdot \frac{t}{T-1}\right\}$ для $T > 1$ и $\sigma(t) = \sigma_0$ для $T = 1$.
3. Инициализация текущего набора активных нейронов $V^+ := \varnothing$.
4. Выполнение шагов 4.1–4.8 для каждого вектора $\boldsymbol{x_k}$ ($k = 1, \ldots, M$).
   4.1. Нормализация весовых коэффициентов нейронов $\boldsymbol{w_{ij}}$ при помощи покомпонентного деления на $\|\boldsymbol{w_{ij}}\|$ ($\boldsymbol{w_{ij}} \neq \vec{\mathbf{0}}$).
   4.2. Нормализация вектора $\boldsymbol{x_k}$ при помощи покомпонентного деления на $\|\boldsymbol{x_k}\|$ ($\boldsymbol{x_k} \neq \vec{\mathbf{0}}$).
   4.3. Вычисление расстояний между вектором $\boldsymbol{x_k}$ и каждым весовым вектором $\boldsymbol{w_{ij}}$ нейрона: $d_{ijk} = D(\boldsymbol{x_k}, \boldsymbol{w_{ij}}) = \sqrt{\sum_{l=1}^{n} (x_{kl} - w_{ijl})^2}$.

4.4. Определение координат нейрона-победителя для вектора $\boldsymbol{x_k}$: $(i_k^*,j_k^*) = F_{IJ}(\boldsymbol{x_k}) = \arg \min\limits_{\substack{1 \leqslant i \leqslant I \\ 1 \leqslant j \leqslant J}} d_{ijk}$.

4.5. Определение текущей окрестности нейрона-победителя (i_k^*,j_k^*): $V^* = V_{(i_k^*,j_k^*)} = \left\{ (i,j) \,\middle|\, \binom{1 \leqslant i \leqslant I}{1 \leqslant j \leqslant J} \wedge (i - i_k^*)^2 + (j - j_k^*)^2 < \sigma^2(t) \right\}$.

4.6. Модификация весовых коэффициентов нейронов с координатами $(i,j) \in V^*$: $\boldsymbol{w_{ij}} \coloneqq \boldsymbol{w_{ij}} + \kappa(t) \cdot \varphi(i,j,t) \cdot (\boldsymbol{x_k} - \boldsymbol{w_{ij}})$, где функция $\varphi(i,j,t) = \exp\left\{ -\frac{(i-i_k^*)^2 + (j-j_k^*)^2}{2 \cdot \sigma^2(t)} \right\}$.

4.7. Расширение набора активных нейронов: $V^+ \coloneqq V^+ \bigcup \{(i_k^*,j_k^*)\}$.

4.8. Применение стратегии генетической оптимизации G' к весам $\boldsymbol{w_{ij}}$ для $(i,j) \in V^\dagger$, где $V^\dagger = V^* \setminus \{(i_k^*,j_k^*)\}$.

5. Применение стратегии генетической оптимизации G' к весам $\boldsymbol{w_{ij}}$ для $(i,j) \in V^\dagger$, где $V^\dagger = V_{(i_M^*,j_M^*)} \setminus V^+$.

6. Увеличение счетчика текущих итераций: $t \coloneqq t + 1$.

7. Переход к следующему шагу (8) при выполнении условия $t \geqslant T$, иначе переход к шагу 2.

8. Исключение «мертвых» нейронов с координатами $(i',j') \in V^\dagger$ на выходной решетке, где $V^\dagger = \{(i,j) | \forall k \in \{1,\ldots,M\} \;\; (i,j) \neq F_{IJ}(\boldsymbol{x_k})\}$.

9. Вычисление порога активации оставшихся нейронов с координатами $(i'',j'') \notin V^\dagger$: $h^-_{i''j''} = \min\limits_{1 \leqslant k \leqslant M} \left\{ d^{-1}_{i''j''k} \,|(i'',j'') = F_{IJ}(\boldsymbol{x_k}) \right\}$.

После модификации весовых коэффициентов при предъявлении обучающего вектора $\boldsymbol{x_k}$ (п. 4.8) либо после выполнения нескольких эпох конкурентного обучения (п. 5) дополнительно применяется основанная на генетическом алгоритме стохастическая оптимизация весовых коэффициентов определенных нейронов выходной решетки карты Кохонена. С этой целью веса каждого нейрона, находящегося в окрестности текущего нейрона-победителя и ни разу не активировавшегося, представляются как последовательность генов, выступающих в роли минимальной единицы для входного аргумента оператора скрещивания (O_1). В результате выполнения этого оператора формируется пара новых хромосом, в которых переставлены местами произвольно выбранные участки ген родительских хромосом (рис. 2.6). Каждый ген является набором битов, который

можно рассматривать как отдельный компонент вектора, ассоциированного с соответствующим нейроном-победителем или одним из нейронов, лежащих в его окрестности. При использовании оператора мутации (O_2) осуществляется перестановка пары случайно выбранных битов внутри одного гена (рис. 2.7), при использовании оператора инверсии (O_3) происходит инвертирование значения случайно выбранного бита (рис. 2.8). Оба этих оператора применяются только к части мантиссы 64-битного „вещественного гена“. Для имитации процесса эволюции нейронов было разработано несколько способов генерации порождаемых ими поколений. Первый подход A_1 заключается в применении операторов скрещивания, мутации или инверсии произвольных нейронов, находящихся в текущей близости от нейрона-победителя. Второй подход A_2 основан на геометрических соображениях о взаимном расположении нейронов на выходной решетке относительно нейрона-победителя: поскольку весовой вектор каждого нейрона, равноотстоящего от нейрона-победителя, модифицируется с одинаковым коэффициентом в результате выполнения выбранного алгоритма обучения, то и оператор скрещивания предлагается применять именно к таким нейронам. Третий подход A_3 подразумевает применение оператора скрещивания только к наиболее приспособленным особям, в то время как к оставшимся нейронам будет применяться оператор мутации или инверсии. Здесь в роли функции приспособленности было выбрано обратное значение величины среднего отклонения при распознавании данным нейроном с весовым коэффициентом $\boldsymbol{w_{ij}}$ элементов обучающей выборки $\{\boldsymbol{x_k}\}_{k=1}^{M}$: $\Psi_{ij} = \left(\frac{\sum_{k=1}^{M}\|\boldsymbol{x_k}-\boldsymbol{w_{ij}}\|}{M}\right)^{-1}$ для п. 5 либо $\Psi_{ijk} = \|\boldsymbol{x_k}-\boldsymbol{w_{ij}}\|^{-1}$ для п. 4.8. Кроме того, было разработано несколько стратегий для выбора того или иного способа генерации нейронов: фиксированный выбор определенного подхода G_1, последовательный или случайный перебор всех подходов G_2 и выбор, основанный на механизме рулетки G_3 [32]. В стратегии G_3 если сгенерированное при помощи выбранного подхода потомство нейронов является более приспособленным по сравнению с предками, то вероятность выбора такого подхода в будущем увеличивается по сравнению с остальными подходами, иначе вероятность его выбора уменьшается.

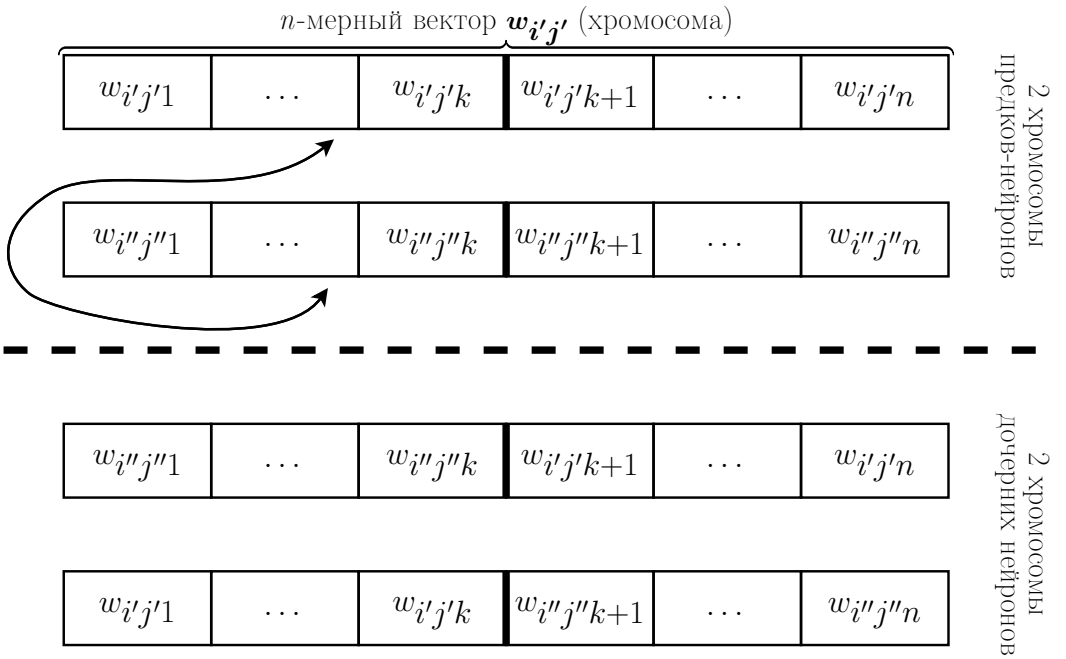


Рисунок 2.6 — Кроссинговер

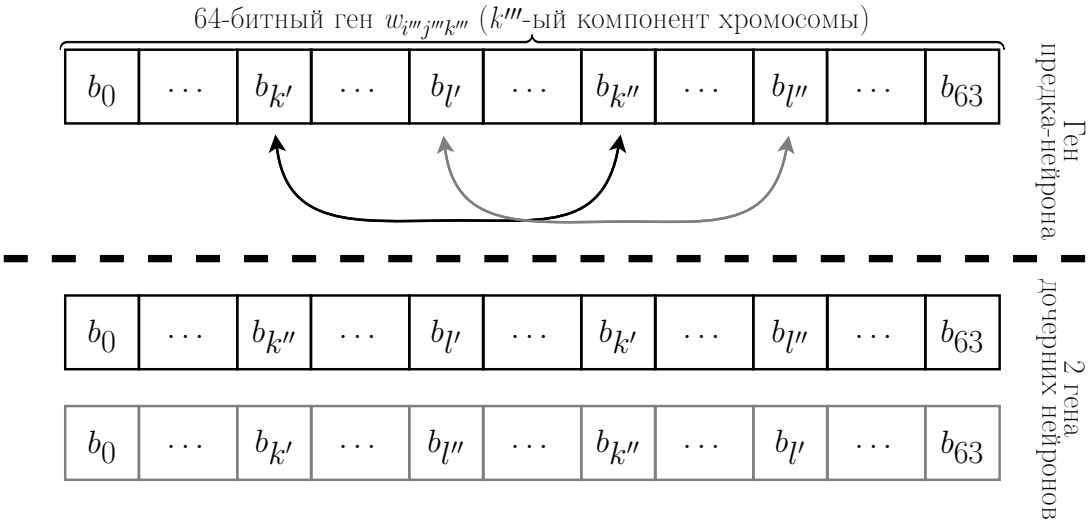


Рисунок 2.7 — Мутация

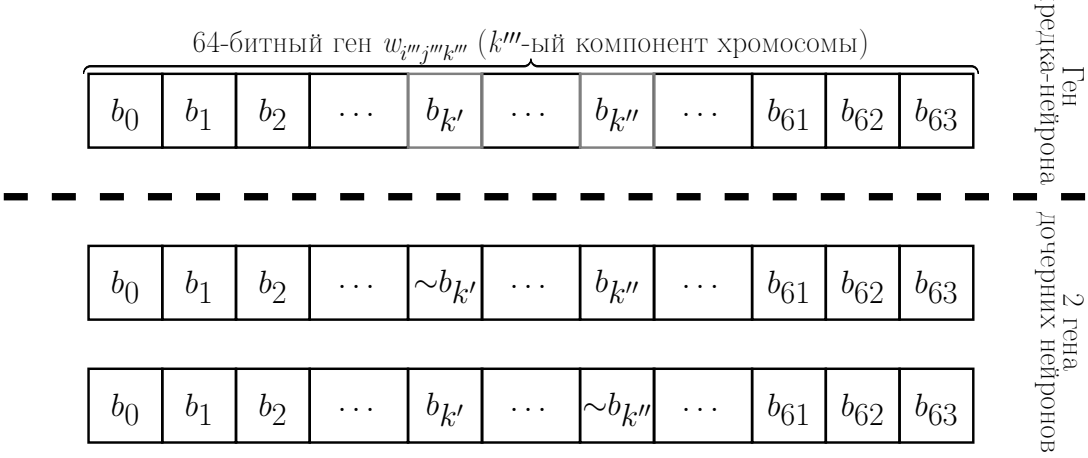


Рисунок 2.8 — Инверсия

Каждый из подходов A_1, A_2, A_3 применяется для создания двух дочерних карт Кохонена, в которых изменениям подвергаются только нейроны из окрестности нейрона-победителя родительской карты Кохонена. Эти подходы могут быть описаны как последовательность следующих действий, в которых различным является только третий шаг: (1) $V \coloneqq \varnothing$; (2) Произвольный выбор оператора O_{i^*} ($i^* \in \{1,2,3\}$); (3_{A_1}) Применение оператора O_{i^*} к очередному нейрону с координатами $(i',j') \in V^\dagger \setminus V$ из текущей окрестности V^* нейрона-победителя (если O_{i^*} — оператор скрещивания, то необходимо дополнительно выбрать еще один нейрон с координатами (i'',j''), отличными от (i',j')); (3_{A_2}) Применение оператора срещивания к паре нейронов с координатами $(i',j') \in V^\dagger \setminus V$ и $(i'',j'') \in U = \left\{(i,j)\right\}_{(i,j)\in V^*\setminus V\setminus\{(i',j')\}\wedge\left(i-i^*_{k(M)}\right)^2+\left(j-j^*_{k(M)}\right)^2=\left(i'-i^*_{k(M)}\right)^2+\left(j'-j^*_{k(M)}\right)^2}$ из окрестности V^* нейрона-победителя с координатами $\left(i^*_{k(M)},j^*_{k(M)}\right)$ (если $U = \varnothing$, то необходимо применить дважды оператор мутации или инверсии к нейрону с координатами (i',j')); (3_{A_3}) Применение оператора скрещивания к нейронам с координатами $(i',j') = \arg\max\limits_{(i,j)\in V^\dagger\setminus V} \Psi_{ij(k)}$ и $(i'',j'') = \arg\max\limits_{(i,j)\in V^*\setminus V\setminus\{(i',j')\}} \Psi_{ij(k)}$ (если $V^\dagger \setminus V \setminus \{(i',j')\} = \varnothing \vee \#\left(V^\dagger \setminus V\right) \leqslant K \cdot \#V^*$, то необходимо применить дважды оператор мутации или инверсии к нейрону с координатами (i',j'), где $0 \leqslant K \leqslant 1$ — константа, определяющая относительное число нейронов, к которым должен применяться только оператор мутации или инверсии); (4) Добавление двух новых сгенерированных при помощи оператора O_{i^*} нейронов по одному в каждую из двух дочерних карт Кохонена с сохранением каждого из этих нейронов в позиции (i',j') на выходной решетке; (5) Добавление координат (i',j') к множеству V: $V \coloneqq V \bigcup \{(i',j')\}$; (6) Переход к шагу 2, если $V \neq V^\dagger$; иначе останов.

Каждая из стратегий G_1, G_2, G_3 манипулирует выбором подходов A_1, A_2, A_3. Стратегия G_1 является самой примитивной из рассматриваемых и заключается в выборе одного из трех подходов A_1, A_2, A_3 на шаге 1 алгоритма генетико-конкурентного обучения сети Кохонена. В дальнейшем на протяжении всего процесса обновления весов нейронов используется только этот подход, обеспечивающий генерацию дочерних карт Кохонена. Стратегия G_2, в отличие

от стратегии G_1, чередует использование подходов либо по круговому принципу, либо рандомизированно. В этой стратегии обеспечивается разнообразие применяемых подходов, однако в ней не учитывается одна их важная характеристика — способность подходов генерировать более приспособленные поколения нейронов. Этого недостатка лишена стратегия G_3, которая накапливает историю „успешности“ подходов и представляется следующим образом: (0) Закрепление за каждым из подходов A_i ($i = 1,2,3$) равновероятной доли его выбора $p_{A_i} = \frac{1}{3}$ ($i = 1,2,3$); (1) Случайный выбор одного из подходов $A_{i_1^*}$ в соответствии со значением $p_{A_{i_1^*}}$ ($i_1^* \in \{1,2,3\}$). С этой целью интервал $[0,1)$ представляется как объединение интервалов $r_{A_1} = [0,p_{A_1})$, $r_{A_2} = [p_{A_1},p_{A_1}+p_{A_2})$, $r_{A_3} = [p_{A_1}+p_{A_2},1)$, и генерируется случайное вещественное число $r \in [0,1)$. Если $r \in r_{A_{i_1^*}}$, то подход $A_{i_1^*}$ является предпочтительным для генерации новых поколений дочерних нейронов V_2^* и V_3^* при помощи родительских нейронов из окрестности V_1^*; (2) Генерация двух дочерних карт Кохонена, в которых нейроны из окрестности V_1^* родительской карты Кохонена заменяются новыми дочерними нейронами (V_2^* и V_3^*) в соответствии с подходом $A_{i_1^*}$; (3) Вычисление усредненных значений функции приспособленности нейронов из окрестностей V_1^*, V_2^* и V_3^* по формуле $\Psi_{V_l^*} = \frac{1}{\#V_l^*} \cdot \sum_{(i,j)\in V_l^*} \Psi_{ij(k)}$ ($l = 1,2,3$). Если сгенерированное потомство обладает бóльшим значением функции приспособленности по сравнению с предком, т.е. если $\max\limits_{l=2,3} \left\{\Psi_{V_l^*}\right\} > \Psi_{V_1^*}$, то нейроны из окрестности V_1^* родительской карты Кохонена заменяются нейронами из окрестности $V_{l^*}^*$ дочерней карты Кохонена, где $l^* = \arg\max\limits_{l=2,3} \left\{\Psi_{V_l^*}\right\}$, и вероятность выбора подхода $A_{i_1^*}$ увеличивается, в отличие от остальных подходов $A_{i_2^*}$, $A_{i_3^*}$ ($i_2^* \in \{1,2,3\} \setminus \{i_1^*\}$, $i_3^* \in \{1,2,3\} \setminus \left(\bigcup_{l=1,2} \{i_l^*\}\right)$), следующим образом: $p_{A_{i_1^*}} := p_{A_{i_1^*}} + \Delta'_{A_{i_1^*}}$, $p_{A_{i_2^*}} := p_{A_{i_2^*}} - \Delta'_{A_{i_2^*}}$, $p_{A_{i_3^*}} := 1 - p_{A_{i_1^*}} - p_{A_{i_2^*}}$, где $\Delta'_{A_{i_1^*}} = \min\left\{\frac{\Psi_{V_{l^*}^*} - \Psi_{V_1^*}}{\Psi_{V_{l^*}^*}}, 1 - p_{A_{i_1^*}}\right\}$, $\Delta'_{A_{i_2^*}} = \min\left\{\frac{1}{2} \cdot \Delta'_{A_{i_1^*}}, p_{A_{i_2^*}}\right\}$. Иначе родительская карта Кохонена остается без изменений, а вероятность выбора подходов $A_{i_1^*}$, $A_{i_2^*}$, $A_{i_3^*}$, применяемая на следующей итерации/эпохе, пересчитывается следующим образом: $p_{A_{i_1^*}} := p_{A_{i_1^*}} - \Delta''_{A_{i_1^*}}$, $p_{A_{i_2^*}} := p_{A_{i_2^*}} + \Delta''_{A_{i_2^*}}$, $p_{A_{i_3^*}} := 1 - p_{A_{i_1^*}} - p_{A_{i_2^*}}$, где $\Delta''_{A_{i_1^*}} = \min\left\{\frac{\Psi_{V_1^*} - \Psi_{V_{l^*}^*}}{\Psi_{V_1^*}}, p_{A_{i_1^*}}\right\}$, $\Delta''_{A_{i_2^*}} = \min\left\{\frac{1}{2} \cdot \Delta''_{A_{i_1^*}}, 1 - p_{A_{i_2^*}}\right\}$. Величина $\frac{\Psi_{V_{l^*}^*} - \Psi_{V_1^*}}{\Psi_{V_{l^*}^*}}$

$(\frac{\Psi_{V_1^*}-\Psi_{V_{l^*}^*}}{\Psi_{V_1^*}})$ показывает относительное изменение приспособленности нового поколения нейронов к обнаружению сетевых атак по сравнению с приспособленностью окружения в родительской карте Кохонена. Использование такой величины позволяет динамически регулировать вес подхода $p_{A_{i_1^*}}$ в зависимости от того, насколько сгенерированное при помощи него поколение является более (менее) приспособленным по сравнению с исходным поколением нейронов. Отметим, что указанный здесь этап 0, включающий первоначальную инициализацию весов $p_{A_{i_1^*}}$, $p_{A_{i_2^*}}$, $p_{A_{i_3^*}}$ равными значениями, выполняется только один раз, к примеру на шаге 1 в алгоритме конкурентного обучения сети Кохонена. Поскольку согласно этой стратегии веса $p_{A_{i_1^*}}$, $p_{A_{i_2^*}}$, $p_{A_{i_3^*}}$, характеризующие способность подходов $A_{i_1^*}$, $A_{i_2^*}$, $A_{i_3^*}$ генерировать поколения более приспособленных нейронов, влияют на выбор того или иного подхода, то можно утверждать, что помимо конкуренции между нейронами (согласно алгоритму конкурентного обучения сети Кохонена) в случае механизма рулетки создается также своеобразная конкуренция между подходами за право генерировать наборы (дочерние поколения) нейронов.

Описанная генетическая оптимизация направлена в первую очередь на уменьшение числа эпох в алгоритме конкурентного обучения сети Кохонена и для достижения условия $\frac{1}{M} \cdot E(\boldsymbol{w_{11}}, \ldots, \boldsymbol{w_{IJ}}) < \varepsilon$ при заданной величине ε.

После обучения на стадии тестирования считается, что нейрон с координатами (i,j) активируется (и тем самым карта Кохонена распознает сетевую атаку) в ответ на входной объект $\boldsymbol{z} = (z_1, \ldots, z_n)^T$ тогда и только тогда, когда $(i,j) = F_{IJ}(\boldsymbol{z}) \wedge \|\boldsymbol{z} - \boldsymbol{w_{ij}}\|^{-1} \geqslant h_{ij}^{-}$. Истинность первого предиката означает, что данный нейрон является нейроном-победителем, в то время как истинность второго предиката отражает достаточную близость входного объекта $\boldsymbol{z}$ к весовому вектору $\boldsymbol{w_{ij}}$ рассматриваемого нейрона. Важно отметить, что используемые здесь обратные величины расстояний необходимы для выполнения одного из главных свойств функции приспособленности — возрастающей монотонности. Ведь чем меньше расстояние между классифицируемым объектом и вектором нейрона-победителя, тем больше их соответствие.

2.4 Модели и алгоритмы обучения бинарных классификаторов

Модель многослойной нейронной сети. По структуре и функционированию искусственные нейронные сети являются аналогом человеческого мозга. Нейронная сеть представляется как многоуровневая модель с вычислительными узлами в качестве нейронов и связями, соединяющими их, в роли синапсов. Моделирование передачи возмущающих нервных импульсов от одного нейрона к другому сводится к установлению связей между элементами рассматриваемой вычислительной структуры. С этой целью каждая такая связь снабжается весовым коэффициентом, чье значение прямо пропорционально выражает значимость входящего сигнала. Чем больше значимость сигнала, тем больший мультипликативный вес назначается соответствующей связи. После настройки подобные структуры, задаваемые как минимум двумя слоями, способны выполнять достаточно точное приближение элементов обучающей выборки [13, 42, 82, 105, 123]: нейронные сети можно рассматривать как универсальные аппроксиматоры, позволяющие описывать и обобщать сложные процессы и явления на основе имеющихся данных, полученных в процессе наблюдения (обучающих данных). На рисунке 2.9 схематически изображен биологический прототип искусственного нейрона. Биологический нейрон состоит из ядра, сомы (тела нейронной клетки) и нескольких разветвляющихся от сомы нервных отростков двух типов [17, 39, 100]. Отросток первого типа, называемый аксоном, представлен в единственном экземпляре в каждом нейроне и является своеобразным передатчиком сгенерированного данным нейроном нервного импульса. Отростки второго типа, именуемые дендритами, принимают поступающие по аксонам соседних клеток сигналы. Участок нервных волокон, находящийся на стыке аксона и дендрита, называется синапсом. В зависимости от типа этот элемент нейрона может вызывать возбуждение или торможение передаваемых сигналов, что при проекции на искусственный аналог нейрона превращается соответственно в усиление или ослабление связи между нейронами.

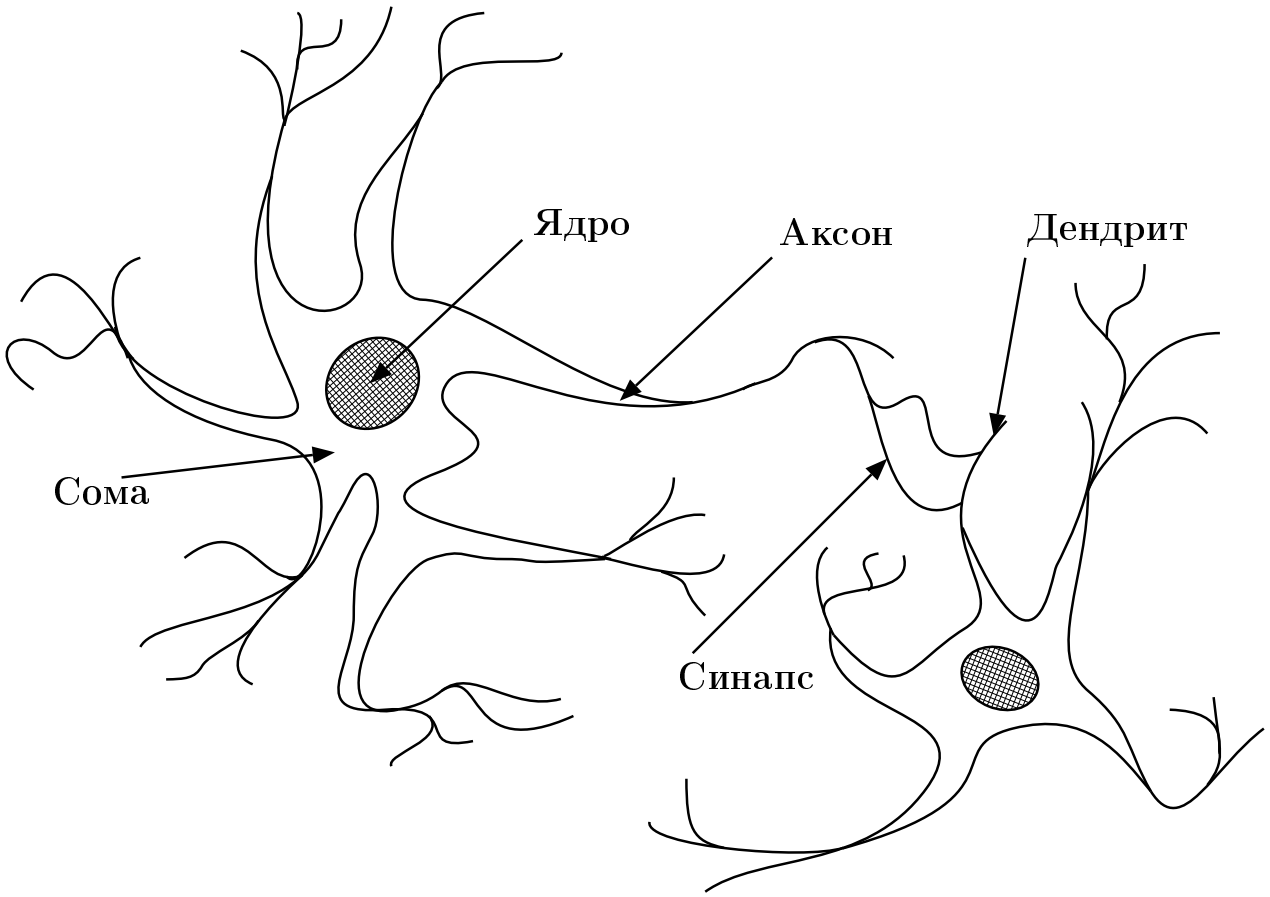


Рисунок 2.9 — Биологический нейрон

Каждый N'-ый слой нейронной сети представляет собой набор нейронов, которые получают сигналы, поступающие от $(N'-1)$-ого уровня, и производят композицию взвешенного суммирования и заданной функции активации. Роль функции активации заключается в сжатии взвешенной суммы сигналов посредством ее приведения к интервалу $(0,1)$ ($[0,1]$) или $(-1,1)$ ($[-1,1]$). Такое последовательное преобразование входного вектора позволяет восстанавливать сложные нелинейные зависимости и обнаруживать сложные скрытые закономерности, которые характерны при решении задач аппроксимации функций, прогнозирования событий, принятия решений и распознавания образов.

В настоящем диссертационном исследовании для классификации сетевых атак разработана модель трехслойной нейронной сети, состоящей из двух скрытых слоев и одного выходного слоя, причем во всех узлах трех слоев используется функция активации одного типа. Схематическое изображение рассматриваемой нейронной сети представлено на рисунке 2.10.

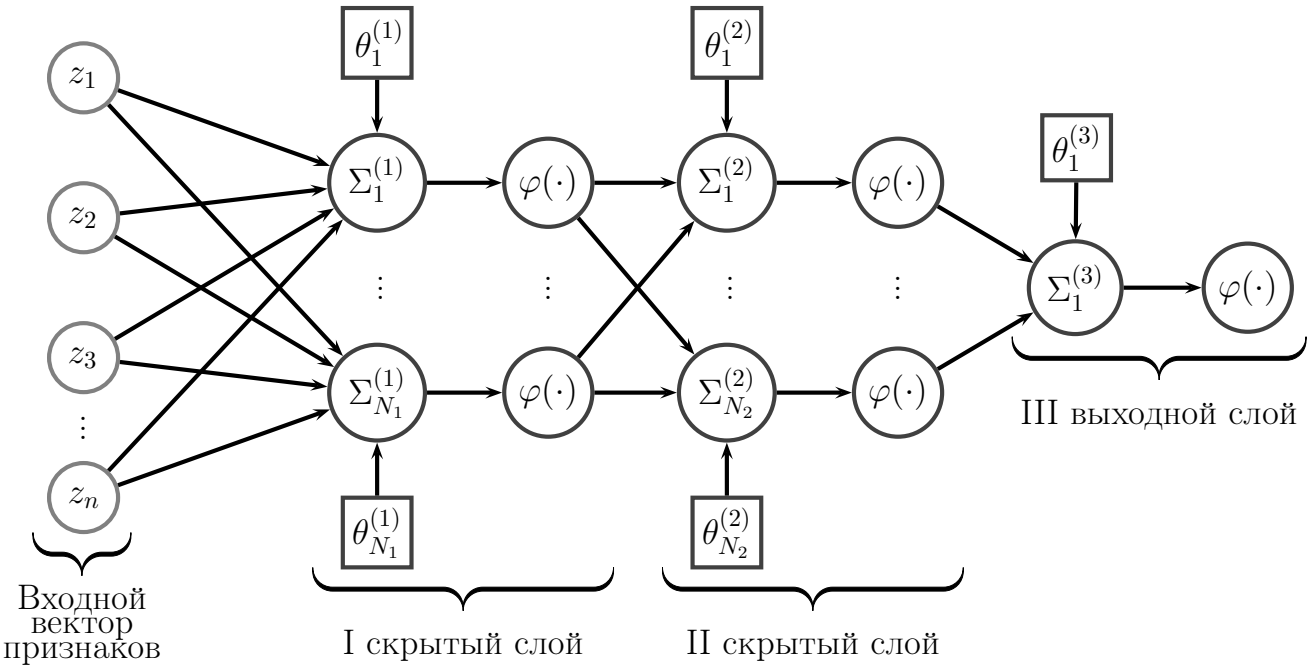


Рисунок 2.10 — Трехслойная нейронная сеть

Выбор такой структуры многослойной нейронной сети (МНС) обусловлен простотой реализации, приемлемой скоростью обучения и возможностью разбиения линейно неразделимых множеств классифицируемых объектов. Именно такой классификатор выступает в роли одного из простейших бинарных детекторов, рассматриваемых в данном исследовании.

Входной слой нейронной сети представляет собой фиктивный слой, который выполняет функцию предварительного распределения поступающих сигналов перед их непосредственной обработкой. Входной вектор каждого узла первого скрытого слоя — это скалярное произведение вектора синаптических весов и входного вектора $\boldsymbol{z} = (z_1, \ldots, z_n)^T$. Сигнал, поступающий на вход i'-ого нейрона первого слоя, состоящего из N_1 узлов, конструируется следующим образом: $z_{i'}^{(1)} = \sum_{j=1}^{n} w_{i'j}^{(1)} \cdot z_j + \theta_{i'}^{(1)}$, где $i' = 1, \ldots, N_1$, $w_{i'j}^{(1)}$ — веса, задающие преобразование сигналов $\boldsymbol{z}$ на входе i'-ого нейрона первого скрытого слоя, $\theta_{i'}^{(1)}$ — параметр смещения i'-ого нейрона, размещенного в первом скрытом слое. Выходным сигналом i'-ого нейрона первого скрытого слоя можно считать величину $y_{i'}^{(1)} = \varphi\left(z_{i'}^{(1)}\right)$. Аналогичным образом задаются входной и выходной сигналы для каждого i''-ого нейрона, который расположен во втором скрытом слое, имеющем N_2 нейронов: $z_{i''}^{(2)} = \sum_{j=1}^{N_1} w_{i''j}^{(2)} \cdot y_j^{(1)} + \theta_{i''}^{(2)}$ и $y_{i''}^{(2)} = \varphi\left(z_{i''}^{(2)}\right)$, где $i'' = 1, \ldots, N_2$,

$w_{i''j}^{(2)}$ — веса, задающие преобразование сигналов $\boldsymbol{y^{(1)}} = (y_1^{(1)}, \ldots, y_{N_1}^{(1)})^T$ на входе i''-ого нейрона второго скрытого слоя, $\theta_{i''}^{(2)}$ — параметр смещения i''-ого нейрона. В роли функции активации φ чаще всего используются нечетная сигмоидальная функция (гиперболический тангенс $\varphi(x) = \frac{\exp\{s \cdot x\} - \exp\{-s \cdot x\}}{\exp\{s \cdot x\} + \exp\{-s \cdot x\}}$), логистическая сигмоидальная функция ($\varphi(x) = (1 + \exp\{-s \cdot x\})^{-1}$), кусочно-линейная функция ($\varphi(x) = \frac{1}{2} \cdot |s \cdot x + 1| - \frac{1}{2} \cdot |s \cdot x - 1|$). Здесь $s > 0$ — это параметр, выбираемый для задания крутости или пологости функции активации.

Последний выходной слой имеет единичную размерность, его единственный нейрон отображает преобразованные сигналы второго скрытого слоя в два класса. Один из этих классов соответствует определенному типу атаки, другой класс — отличным от нее соединениям. Результирующий сигнал $y_1^{(3)}$ составляется следующим образом: $y_1^{(3)} = \varphi\left(\sum_{j=1}^{N_2} w_{1j}^{(3)} \cdot y_j^{(2)} + \theta_1^{(3)}\right)$, где $w_{1j}^{(3)}$ — веса на входе нейрона последнего слоя, $\theta_1^{(3)}$ — параметр смещения выходного нейрона.

Функционирование нейросетевой классификационной модели может быть описано следующей формулой:

$$Y(\boldsymbol{z}) = \varphi\left(\sum_{i=1}^{N_2} w_{1i}^{(3)} \cdot \varphi\left(\sum_{j=1}^{N_1} w_{ij}^{(2)} \cdot \varphi\left(\sum_{k=1}^{n} w_{jk}^{(1)} \cdot z_k + \theta_j^{(1)}\right) + \theta_i^{(2)}\right) + \theta_1^{(3)}\right).$$

Рассмотрим алгоритм обратного распространения ошибки [9, 204], являющийся наиболее популярным алгоритмом обучения МНС (алгоритм 7).

Алгоритм 7: Алгоритм обучения многослойной нейронной сети

---

1. Задание структуры нейронной сети (выбор числа скрытых слоев и нейронов, расположенных в них, типов функций активации).
2. Задание максимального числа эпох обучения T и минимального значения суммарной среднеквадратичной ошибки ε.
3. Зануление счетчика текущих итераций $t := 0$ и инициализация весовых коэффициентов $w_{ij}^{(K)}$ произвольными значениями, где K обозначает номер слоя, i соответствует номеру позиции нейрона в K-ом слое, j отображает наличие связи между текущим нейроном и выходным сиг-

налом j-ого нейрона в $(K-1)$-ом слое, если $K > 1$, или j-ым входным сигналом, если $K = 1$.

4. Выполнение шагов 4.1–4.3 для каждого вектора $\boldsymbol{x_k}$ ($k = 1, \ldots, M$).
   4.1. Прямое распространение сигналов: вычисление входящих сигналов для каждого i-ого нейрона, расположенного в K-ом слое, по формуле: $x_i^{(K)} = \sum_{j=1}^{N_{K-1}+1} w_{ij}^{(K)} \cdot y_j^{(K-1)}$, где N_{K-1} — число нейронов в $(K-1)$-ом слое, $w_{ij}^{(K)} = \theta_i^{(K)}$ и $y_j^{(K-1)} = 1$ для $j = N_{K-1} + 1$, $y_j^{(K-1)} = \varphi\left(x_j^{(K-1)}\right)$ для $K > 1$ и $y_j^{(K-1)} = x_{kj}$ (исходный сигнал) для $K = 1$.
   4.2. Обратное распространение ошибки: вычисление приращений весовых коэффициентов нейронов по формуле: $\Delta w_{ij}^{(K)} = \alpha \cdot \delta_i^{(K)} \cdot y_j^{(K-1)}$, последовательно начиная с последнего слоя и заканчивая первым ($0 < \alpha \leqslant 1$ — коэффициент пропорциональности коррекции весов, известный также как норма обучения [12], или параметр скорости обучения [39]. Если K-ый слой является выходным, то $\delta_i^{(K)} = \varphi'\left(x_i^{(K)}\right) \cdot \left(u_{ki} - y_i^{(K)}\right)$, в противном случае $\delta_i^{(K)} = \varphi'\left(x_i^{(K)}\right) \cdot \sum_{j=1}^{N_{K+1}+1} \delta_j^{(K+1)} \cdot w_{ji}^{(K+1)}$, где u_{ki} обозначает желаемый выход нейронной сети в i-ом нейроне на выходном слое для k-ого примера из обучающей выборки.
   4.3. Корректировка весовых коэффициентов нейронов по формуле: $w_{ij}^{(K)} := w_{ij}^{(K)} + \Delta w_{ij}^{(K)}$.
5. Увеличение счетчика текущих итераций $t := t + 1$.
6. Останов алгоритма при выполнении одного из условий: $t \geqslant T$ или $\sum_{k=1}^{M} E(\boldsymbol{x_k}) \leqslant \varepsilon$, где $E(\boldsymbol{x_k}) = \frac{1}{2} \cdot \sum_{i=1}^{N_{K_{all}}} \left(u_{ki} - y_i^{(K_{all})}\right)^2$ — суммарная среднеквадратичная ошибка нейронной сети, имеющей выходное значение $\boldsymbol{y^{(K_{all})}} = \left(y_1^{(K_{all})}, \ldots, y_{N_{K_{all}}}^{(K_{all})}\right)^T$ при подаче на ее вход вектора $\boldsymbol{x_k}$ (с ожидаемым выходным вектором $\boldsymbol{u_k} = \left(u_{k1}, \ldots, u_{kN_{K_{all}}}\right)^T$) и состоящей из $K_{all} = 3$ слоев и $N_{K_{all}} = 1$ нейронов на выходном слое; в противном случае переход к шагу 4.

Приведенный выше алгоритм принадлежит к общему семейству алгоритмов градиентного спуска, в которых поиск точки минимума осуществляется в направлении, противоположном градиенту оптимизируемой функции (например, среднеквадратичной ошибки). Для таких алгоритмов характерно „проваливание в яму локального минимума“, когда алгоритм практически прекращает модифицировать весовые параметры, несмотря на наличие более глубокого экстремума по сравнению с уже найденным. Эти проблемы частично решаются при помощи различных улучшений алгоритма обратного распространения ошибки, которые могут использовать переменный коэффициент пропорциональности коррекции весов в зависимости от сохранения/изменения знака производной [24, 182], принимать во внимание факторы момента для изменения каждого отдельного веса [24, 99] или учитывать значения вторых производных [145, 152, 157].

Модель нейронной сети с радиальными базисными функциями. Нейронные сети с радиальными базисными функциями (РБФ) построены отличным от рассмотренных выше многослойных нейронных сетей способом. Первый скрытый слой таких нейронных сетей предназначен для проецирования входного вектора $\boldsymbol{z} = (z_1, \ldots, z_n)^T$ в новое пространство признаков $\left(\Phi\left(\boldsymbol{z} - \boldsymbol{w}_1^{(1)}\right), \ldots, \Phi\left(\boldsymbol{z} - \boldsymbol{w}_{N_1}^{(1)}\right)\right)^T$, каждый из которых отражает степень близости входного вектора $\boldsymbol{z}$ и весового вектора $\boldsymbol{w}_i^{(1)} = \left(w_{i1}^{(1)}, \ldots, w_{in}^{(1)}\right)^T$, приписанного i-ому нейрону первого скрытого слоя сети, где $i = 1, \ldots, N_1$. Здесь $\Phi\left(\boldsymbol{z} - \boldsymbol{w}_i^{(1)}\right) = \exp\left\{-\frac{\|\boldsymbol{z} - \boldsymbol{w}_i^{(1)}\|^2}{s}\right\} = \exp\left\{-\frac{\sum_{j=1}^{n}\left(z_j - w_{ij}^{(1)}\right)^2}{s}\right\}$ — радиальная базисная функция. Существенной особенностью является то, что веса этого слоя не изменяются в процессе обучения, а задаются статическими при помощи одного из следующих способов: (1) случайной инициализации, (2) инициализации несколькими случайно выбранными векторами обучающей выборки, (3) инициализации с использованием метода кластеризации. В данном исследовании для инициализации весовых коэффициентов скрытого слоя использовался третий подход, а именно метод K-средних, где выбранное число кластеров совпадало с размерностью скрытого слоя N_1.

Модель нейронной сети с радиальными базисными функциями представляется следующим образом:

$$Y(\boldsymbol{z}) = \varphi\left(\sum_{i=1}^{N_1} w_{1i}^{(2)} \cdot \Phi\left(\boldsymbol{z} - \boldsymbol{w_i^{(1)}}\right) + \theta_1^{(2)}\right).$$

Алгоритм обучения нейронной сети с РБФ представлен в алгоритме 8.

Алгоритм 8: Алгоритм обучения нейронной сети с РБФ

---

1. Задание структуры нейронной сети (выбор размерности скрытого слоя N_1, типа функции активации φ выходного слоя).
2. Задание максимального числа эпох обучения T и минимального значения суммарной среднеквадратичной ошибки ε.
3. Вычисление центроидов $\{\bar{\boldsymbol{v}}_{\boldsymbol{i}}\}_{i=1}^{N_1}$ кластеров при помощи метода K-средних и инициализация весовых коэффициентов $w_{ij}^{(1)}$ нейронов первого скрытого слоя компонентами этих центроидов $w_{ij}^{(1)} \coloneqq \bar{v}_{ij}$ ($i = 1, \ldots, N_1$, $j = 1, \ldots, n$).
4. Зануление счетчика текущих итераций $t \coloneqq 0$ и инициализация весовых коэффициентов $w_{ij}^{(2)}$ нейронов второго выходного слоя произвольными значениями ($i = 1, \ldots, N_2$, $j = 1, \ldots, N_1$).
5. Выполнение шагов 5.1–5.2 для каждого вектора $\boldsymbol{x_k}$ ($k = 1, \ldots, M$).
   5.1. Прямое распространение сигналов: вычисление выходных сигналов для каждого i-ого нейрона, расположенного во втором слое, по формуле: $y_i^{(2)} = \varphi\left(x_i^{(2)}\right)$, где $x_i^{(2)} = \sum_{j=1}^{N_1+1} w_{ij}^{(2)} \cdot y_j^{(1)}$; $w_{ij}^{(2)} = \theta_i^{(2)}$ и $y_j^{(1)} = 1$ для $j = N_1 + 1$; $y_j^{(1)} = \Phi\left(\boldsymbol{x_k} - \boldsymbol{w_j^{(1)}}\right)$ для $j = 1, \ldots, N_1$.
   5.2. Обновление весовых коэффициентов выходного слоя нейронной сети по правилу Видроу–Хоффа [12]: $w_{ij}^{(2)} \coloneqq w_{ij}^{(2)} + \Delta w_{ij}^{(2)}$, где $\Delta w_{ij}^{(2)} = \alpha \cdot \varphi'\left(x_i^{(2)}\right) \cdot \left(u_{ki} - y_i^{(2)}\right) \cdot y_j^{(1)}$, u_{ki} обозначает желаемый выход нейронной сети в i-ом нейроне на выходном слое для k-ого примера из обучающей выборки.
6. Увеличение счетчика текущих итераций $t \coloneqq t + 1$.

7. Останов алгоритма при выполнении одного из условий: $t \geqslant T$ или $\sum_{k=1}^{M} E(\boldsymbol{x_k}) \leqslant \varepsilon$, где $E(\boldsymbol{x_k}) = \frac{1}{2} \cdot \sum_{i=1}^{N_{K_{all}}} \left(u_{ki} - y_i^{(K_{all})}\right)^2$ — суммарная среднеквадратичная ошибка нейронной сети, имеющей выходное значение $\boldsymbol{y^{(K_{all})}} = \left(y_1^{(K_{all})}, \ldots, y_{N_{K_{all}}}^{(K_{all})}\right)^T$ при подаче на ее вход вектора $\boldsymbol{x_k}$ (с ожидаемым выходным вектором $\boldsymbol{u_k} = \left(u_{k1}, \ldots, u_{kN_{K_{all}}}\right)^T$) и состоящей из $K_{all} = 2$ слоев и $N_{K_{all}} = 1$ нейронов на выходном слое; в противном случае переход к шагу 5.

Модель рекуррентной нейронной сети Джордана. Модель рекуррентной нейронной сети (РНС) Джордана представляет собой расширение описанной выше модели МНС при помощи добавления обратной связи типа выход-вход в так называемом контекстном слое.

$$Y(\boldsymbol{z}) = \varphi\left(\sum_{i=1}^{N_1} w_{1i}^{(2)} \cdot \varphi\left(\sum_{j=1}^{n} w_{ij}^{(1)} \cdot z_j + w_{i0}^{(1)} \cdot z_0 + \theta_i^{(1)}\right) + \theta_1^{(2)}\right).$$

Выходной сигнал z_0, полученный в результате анализа одного из предыдущих векторов, сохраняется на несколько тактов перед обработкой нового вектора, дополненного значением этого сигнала, что позволяет нейронной сети запоминать некоторую историю чередования образов аномальных и нормальных сетевых соединений. В остальном принцип функционирования и обучения такой сети (алгоритм 9) является схожим с обучением многослойной нейронной сети.

Алгоритм 9: Алгоритм обучения рекуррентной нейронной сети

---

1. Инициализация значения $z_0 := 0$.
2. Выполнение шагов 1–6 алгоритма 7 с сохранением выходного значения РНС в переменной z_0 после каждой итерации алгоритма.

Модель нейронечеткой сети. Следующим подходом, используемым в настоящем диссертационном исследовании при построении интеллектуального ядра для обнаружения сетевых атак, являются нейронечеткие сети (ННС), являющиеся частным случаем систем нечеткого вывода, которые отражают способ-

ность человеческого мышления принимать решения в условиях неопределенности и нечеткости [38]. Как правило, такие системы состоят из пяти функциональных блоков [126]:

1. Первый блок — это база правил, которая включает набор нечетких импликаций (правил) вида if A then B. Левая часть A такого правила называется посылкой, правая часть B — заключением. Существенным отличием таких правил от традиционных продукционных правил является то, что каждому из утверждений, входящих в состав частей A и B, приписывается некоторое число от 0 до 1, отражающее степень достоверности посылки и заключения.

2. Второй блок — это база данных, содержащая набор функций принадлежности. Эти функции задают для входных лингвистических переменных переход от их количественных (crisp) значений к нечетким (fuzzy) лингвистическим термам. Для каждого из таких термов строится отдельная функция принадлежности (membership function), выходное значение которой характеризует меру принадлежности входной переменной соответствующему нечеткому множеству (терму). Наиболее часто используемым типом функций принадлежности являются непрерывные кусочно-дифференцируемые (треугольные и трапецеидальные функции) или гладкие функции (семейство колоколообразных функций) с областью значений $[0, 1]$ (или $(0, 1]$).

3. Третий блок — это блок фаззификации (введения нечеткости), роль которого заключается в применении к входному аргументу заданной функции принадлежности соответствующего ей лингвистического терма. Каждое из подусловий A_i, входящих в состав конъюнктивной посылки $A = A_1 \wedge \cdots \wedge A_n$, и заключение B представляются в виде нечетких утверждений z_i is γ_i и y is Γ соответственно, где z_i и y — лингвистические переменные, γ_i и Γ — лингвистические термы ($i = 1, \ldots, n$). Здесь рассматриваются посылки, в которых нечеткие утверждения объединены с использованием только одного логического оператора „и“, хотя в общем случае допускаются и другие логические операторы. Результатом этапа фаззификации является набор вычисленных значений этих нечетких утверждений.

4. Четвертый блок — это блок нечеткого вывода, содержащий набор уже встроенных в его ядро нечетких импликаций и предоставляющий механизм (к примеру, правило modus ponens или modus tollens [32]) для вычисления заключения B по входному набору конъюнкций в части посылки A. Для вычисления полной степени истинности левой части применяются Т-нормы, наиболее распространенными примерами которых являются операции минимума и произведения [32]. На выходе блока нечеткого вывода для лингвистической переменной y формируется один или несколько нечетких термов вместе с соответствующими для них значениями функций принадлежности.

5. Пятый блок — это блок дефаззификации (приведения к четкости), восстанавливающий количественное значение лингвистической переменной y по ее нечетким значениям. А именно, полученные в результате работы блока нечеткого вывода данные преобразуются в количественные значения при помощи одного из следующих методов: метода центра площади, метода центра тяжести [18], метода суммы центров, метода максимума функции принадлежности [32] и т.д.

На рисунке 2.11 изображены блоки системы нечеткого вывода с указанной в виде стрелок очередностью взаимодействия между ними.

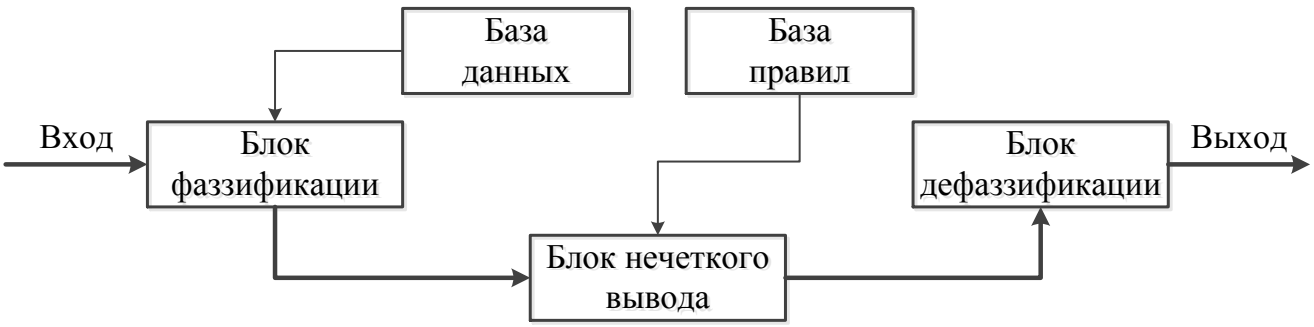


Рисунок 2.11 — Блоки системы нечеткого вывода

В описанной системе нечеткого вывода заключения B во всех правилах if A then B имели вид нечеткого утверждения y is Γ, которое не зависит от лингвистических переменных, входящих в состав посылки A. Подход, предложенный Такаги и Сугено [195], направлен на устранение этого недостатка и заключается во введении в правую часть каждого из правил некоторой функциональной зависимости от элементов его левой части, а именно $y = f(z_1, \ldots, z_n)$. В ситуа-

циях, приближенных к реальным жизненным, часто приходится сталкиваться с моделями подобного типа, в частности, когда человек или устройство не имеет возможности точно оценить величины входных параметров, но при этом регулирующее воздействие может быть явно вычислено по известной формуле.

Нейронечеткая сеть (Adaptive-Network-Based Fuzzy Inference System, ANFIS) [126] является развитием модели Такаги–Сугено, в которую добавлен элемент адаптивной настройки (обучения) ее параметров. Такая сеть состоит из пяти слоев, где входной сигнал претерпевает изменения, распространяясь последовательно от первого до последнего слоя. В такой сети в правой части каждого правила используется полиномиальная зависимость от входных лингвистических переменных. Поэтому каждое нечеткое правило в такой сети представляется как элемент, принадлежащий набору правил вида:

$$\left\{\text{if } \left(z_1 \text{ is } \gamma_1^{(j_1)} \wedge \cdots \wedge z_n \text{ is } \gamma_n^{(j_n)}\right) \text{ then } \right.$$
$$\left. y = f^{(j)}(z_1,\ldots,z_n) = p_0^{(j)} + p_1^{(j)} \cdot z_1 + \cdots + p_n^{(j)} \cdot z_n \right\}_{j=1}^{Q},$$

где Q обозначает мощность набора нечетких правил, в которых каждая переменная $z_1,\ldots,z_n$ имеет ровно r нечетких термов; $j_1,\ldots,j_n$ обозначают номера нечетких термов, соответствующих лингвистическим переменным $z_1,\ldots,z_n$, в нечетком правиле под номером j ($1 \leqslant j_1 \leqslant r,\ldots,1 \leqslant j_n \leqslant r$). Как и в классической системе нечеткого вывода, левая часть такого правила является конъюнкцией нечетких утверждений, которые выражают степень соответствия входного количественного значения z_i тому или иному лингвистическому терму $\gamma_i^{(j_i)}$ согласно выражению $\mu_{\gamma_i^{(j_i)}}(z_i)$, где в качестве функции принадлежности $\mu_{\gamma_i^{(j_i)}}(z_i)$ чаще всего используются колоколообразная функция

$$\mu_{\gamma_i^{(j_i)}}(z_i) = \left(1 + \left|\frac{z_i - c_{ij}}{a_{ij}}\right|^{2\cdot b_{ij}}\right)^{-1}$$

или гауссова функция

$$\mu_{\gamma_i^{(j_i)}}(z_i) = \exp\left\{-\left(\frac{z_i - c_{ij}}{a_{ij}}\right)^2\right\},$$

где $i = 1,\ldots,n$ и $j = 1,\ldots,Q$.

На рисунке 2.12 изображена нейронечеткая сеть с n лингвистическими переменными и $r = 5$ лингвистическими термами для каждой переменной.

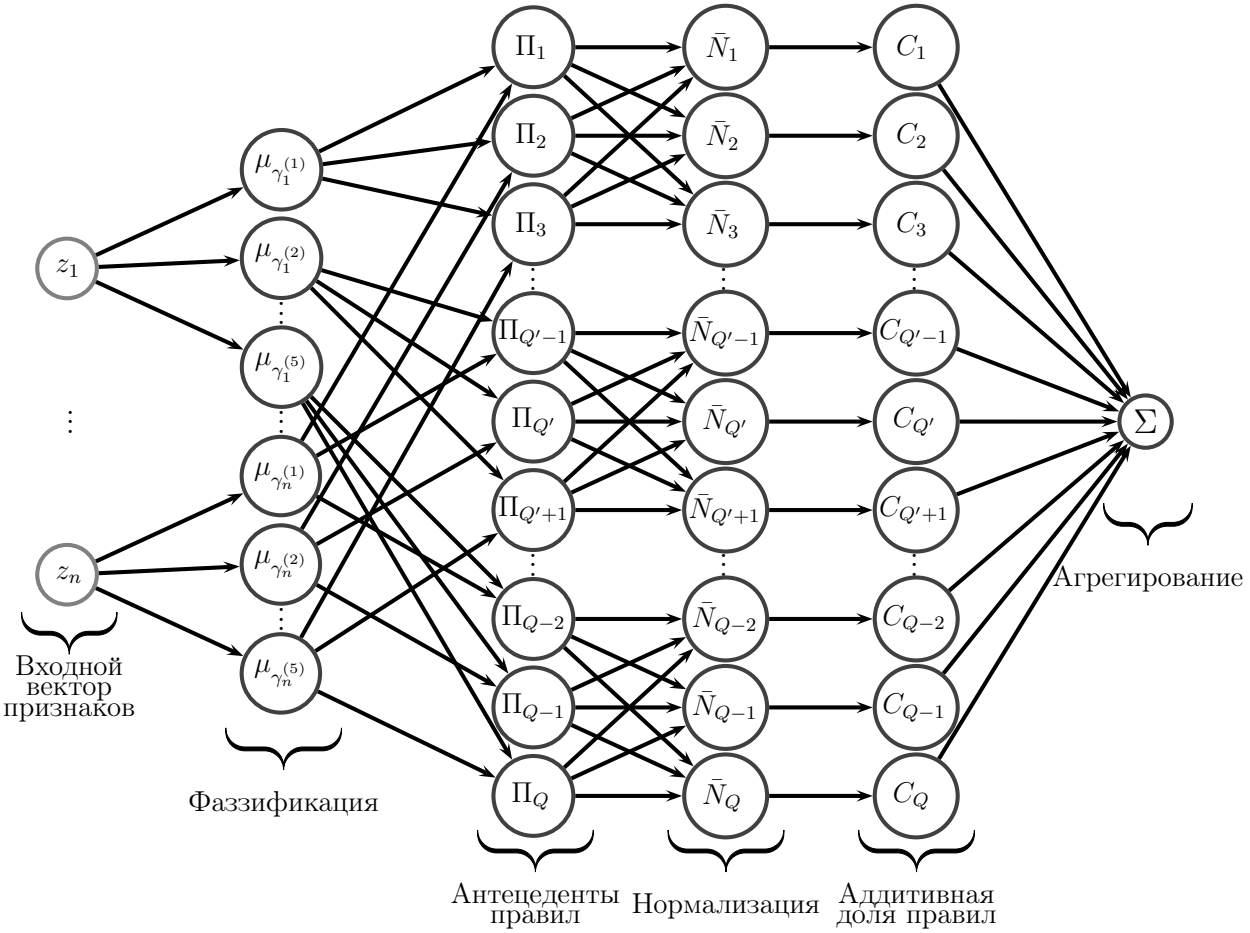


Рисунок 2.12 — Нейронечеткая сеть

Узловые элементы первого слоя в нейронечеткой сети выполняют роль фаззификации входной лингвистической переменной $z_{i'}$, и выходом этого слоя являются значения функции принадлежности $\mu_{\gamma_{i'}^{(j')}}$ этой переменной j'-ому нечеткому множеству $\gamma_{i'}^{(j')}$: $Y_{j'i'}^{(1)} = \mu_{\gamma_{i'}^{(j')}}(z_{i'})$, где $i' = 1,\ldots,n$, $j' = 1,\ldots,r$. Во втором слое осуществляется формирование посылок нечетких правил с их объединением при помощи операции взятия Т-нормы — произведения; k-ое выходное значение в этом слое можно рассматривать как вес, назначенный k-ому правилу: $Y_k^{(2)} = Y_{k_1 1}^{(1)} \times \cdots \times Y_{k_n n}^{(1)} = \mu_{\gamma_1^{(k_1)}}(z_1) \times \cdots \times \mu_{\gamma_n^{(k_n)}}(z_n)$, где $k = 1,\ldots,Q$, причем $Q \leqslant r^n$ при условии отсутствия каких-либо противоречащих друг другу правил. В элементах третьего слоя вычисляется отношение веса соответствующего правила к общей сумме весов всех правил, выходом этого слоя является нормализованная к $[0,1]$ величина: $Y_k^{(3)} = \frac{Y_k^{(2)}}{\sum_{i=1}^{Q} Y_i^{(2)}}$. В четвертом слое

вычисляется результат заключения каждого из правил с учетом полученной на третьем слое относительной степени его выполнения; k-ое выходное значение в этом слое отражает аддитивную долю k-ого правила в общем выходе сети: $Y_k^{(4)} = Y_k^{(3)} \cdot f^{(k)}(z_1, \ldots, z_n) = Y_k^{(3)} \cdot \left(p_0^{(k)} + p_1^{(k)} \cdot z_1 + \cdots + p_n^{(k)} \cdot z_n\right)$. На выходном пятом слое располагается единственный нейрон, отвечающий за суммирование входных сигналов, поступающих от узлов четвертого слоя: $Y^{(5)} = \sum_{i=1}^{Q} Y_i^{(4)} = \sum_{i=1}^{Q} Y_i^{(3)} \cdot f^{(i)}(z_1, \ldots, z_n) = \frac{\sum_{i=1}^{Q} Y_i^{(2)} \cdot f^{(i)}(z_1, \ldots, z_n)}{\sum_{i=1}^{Q} Y_i^{(2)}}$.

Тем самым модель ANFIS представляется при помощи следующего соотношения [4]:

$$Y(\boldsymbol{z}) = \frac{\sum\limits_{i=1}^{Q} \left(\mu_{\gamma_1^{(i_1)}}(z_1) \times \cdots \times \mu_{\gamma_n^{(i_n)}}(z_n)\right) \cdot \left(p_0^{(i)} + p_1^{(i)} \cdot z_1 + \cdots + p_n^{(i)} \cdot z_n\right)}{\sum\limits_{i=1}^{Q} \mu_{\gamma_1^{(i_1)}}(z_1) \times \cdots \times \mu_{\gamma_n^{(i_n)}}(z_n)}.$$

Представленная модель позволяет описывать правила в виде нечетких утверждений и на основе этих аксиом генерировать новые знания, которые могут быть использованы для обнаружения новых и ранее не встречавшихся вариаций сетевых атак. Для обучения такой модели могут применяться как стандартный метод градиентного спуска, так и его модификации. Так, с целью оптимизации алгоритма обратного распространения ошибки автор системы ANFIS [126] предлагает использовать гибридное правило ее обучения, которое совмещает в себе метод градиентного спуска и метод наименьших квадратов. Исходное множество настраиваемых параметров a_{ij},b_{ij},c_{ij},$p_0^{(j)}$,$p_1^{(j)}$,$\ldots$,$p_n^{(j)}$ декомпозируется на два подмножества, элементы одного из которых обновляются при помощи метода градиентного спуска, а элементы другого определяются при помощи метода наименьших квадратов. Рассмотрим гибридное правило обучения нейронечеткой сети, предложенное в [126] (алгоритм 10).

Алгоритм 10: Алгоритм обучения нейронечеткой сети

---

1. Задание структуры нейронечеткой сети (выбор числа лингвистических термов r, типа функций принадежности $\mu_{\gamma_i^{(j)}}$).

2. Задание максимального числа эпох обучения T и минимального значения суммарной среднеквадратичной ошибки ε.
3. Зануление счетчика текущих итераций $t \coloneqq 0$ и инициализация параметров a_{ij},b_{ij},c_{ij},$p_0^{(j)}$,$p_1^{(j)}$,...,$p_n^{(j)}$ произвольными значениями ($i = 1,\ldots,n$, $j = 1,\ldots,Q$).
4. Вычисление коэффициентов $p_0^{(j)}$,$p_1^{(j)}$,...,$p_n^{(j)}$ по методу наименьших квадратов [4].

Подставляя в формулу модели элементы обучающей выборки $\left\{\boldsymbol{x_i} = \{x_{ij}\}_{j=1}^{n}\right\}_{i=1}^{M}$, получаем следующую систему уравнений:

$$\underbrace{\begin{pmatrix} e_{11} & e_{11}x_{11} & \ldots & e_{11}x_{1n} & \ldots & e_{1Q} & e_{1Q}x_{11} & \ldots & e_{1Q}x_{1n} \\ e_{21} & e_{21}x_{21} & \ldots & e_{21}x_{2n} & \ldots & e_{2Q} & e_{2Q}x_{21} & \ldots & e_{2Q}x_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ e_{M1} & e_{M1}x_{M1} & \ldots & e_{M1}x_{Mn} & \ldots & e_{MQ} & e_{MQ}x_{M1} & \ldots & e_{MQ}x_{Mn} \end{pmatrix}}_{\boldsymbol{H}} \cdot \underbrace{\begin{pmatrix} p_0^{(1)} \\ p_1^{(1)} \\ \vdots \\ p_n^{(1)} \\ \vdots \\ p_0^{(Q)} \\ p_1^{(Q)} \\ \vdots \\ p_n^{(Q)} \end{pmatrix}}_{\boldsymbol{p}} =$$

$$= \underbrace{\begin{pmatrix} u_1 \\ \vdots \\ u_M \end{pmatrix}}_{\boldsymbol{u}},$$

где

$$e_{ij} = \frac{\mu_{\gamma_1^{(j_1)}}(x_{i1}) \times \cdots \times \mu_{\gamma_n^{(j_n)}}(x_{in})}{\sum\limits_{k=1}^{Q} \mu_{\gamma_1^{(k_1)}}(x_{i1}) \times \cdots \times \mu_{\gamma_n^{(k_n)}}(x_{in})}.$$

Как правило, $M \gg (n+1)\cdot Q$, поэтому данная система уравнений может не иметь решений. В этом случае прибегают к приему нахождения такого вектора $\boldsymbol{p}$, который удовлетворяет следующему условию [36]:

$$\xi(\boldsymbol{p}) = \|\boldsymbol{H}\cdot\boldsymbol{p}-\boldsymbol{u}\|^2 = (\boldsymbol{H}\cdot\boldsymbol{p}-\boldsymbol{u})^T\cdot(\boldsymbol{H}\cdot\boldsymbol{p}-\boldsymbol{u}) \to \min$$

$$\begin{aligned}\xi(\boldsymbol{p}) &= (\boldsymbol{H}\cdot\boldsymbol{p}-\boldsymbol{u})^T\cdot(\boldsymbol{H}\cdot\boldsymbol{p}-\boldsymbol{u}) = \\ &= (\boldsymbol{H}\cdot\boldsymbol{p})^T\cdot(\boldsymbol{H}\cdot\boldsymbol{p}) - (\boldsymbol{H}\cdot\boldsymbol{p})^T\cdot\boldsymbol{u} - \boldsymbol{u}^T\cdot(\boldsymbol{H}\cdot\boldsymbol{p}) + \boldsymbol{u}^T\cdot\boldsymbol{u} = \\ &= \boldsymbol{p}^T\cdot\boldsymbol{H}^T\cdot\boldsymbol{H}\cdot\boldsymbol{p} - 2\cdot\boldsymbol{u}^T\cdot\boldsymbol{H}\cdot\boldsymbol{p} + \|\boldsymbol{u}\|^2\end{aligned}$$

$$\frac{d\xi(\boldsymbol{p})}{d\boldsymbol{p}} = 2\cdot\boldsymbol{H}^T\cdot\boldsymbol{H}\cdot\boldsymbol{p} - 2\cdot\boldsymbol{u}^T\cdot\boldsymbol{H} = 2\cdot\boldsymbol{H}^T\cdot\boldsymbol{H}\cdot\boldsymbol{p} - 2\cdot\boldsymbol{H}^T\cdot\boldsymbol{u} = \boldsymbol{0}$$

$$\boldsymbol{H}^T\cdot\boldsymbol{H}\cdot\boldsymbol{p} = \boldsymbol{H}^T\cdot\boldsymbol{u} \Rightarrow \boldsymbol{p} = \left(\boldsymbol{H}^T\cdot\boldsymbol{H}\right)^{-1}\cdot\boldsymbol{H}^T\cdot\boldsymbol{u}$$

5. Вычисление коэффициентов a_{ij},b_{ij},c_{ij} по методу обратного распространения ошибки в пакетном режиме [39].

$$a_{ij} \coloneqq a_{ij} - \frac{\alpha_a}{M}\cdot\frac{\partial\sum_{k=1}^{M}E(\boldsymbol{x_k})}{\partial a_{ij}},$$

$$b_{ij} \coloneqq b_{ij} - \frac{\alpha_b}{M}\cdot\frac{\partial\sum_{k=1}^{M}E(\boldsymbol{x_k})}{\partial b_{ij}},$$

$$c_{ij} \coloneqq c_{ij} - \frac{\alpha_c}{M}\cdot\frac{\partial\sum_{k=1}^{M}E(\boldsymbol{x_k})}{\partial c_{ij}},$$

$$0 < \alpha_a \leqslant 1,\ 0 < \alpha_b \leqslant 1,\ 0 < \alpha_c \leqslant 1.$$

6. Увеличение счетчика текущих итераций $t \coloneqq t+1$.
7. Останов алгоритма при выполнении одного из условий: $t \geqslant T$ или $\sum_{k=1}^{M}E(\boldsymbol{x_k}) \leqslant \varepsilon$, где $E(\boldsymbol{x_k}) = \frac{1}{2}\cdot\left(u_k - y_k^{(K_{all})}\right)^2$ — среднеквадратическая ошибка нейронечеткой сети, имеющей $K_{all} = 5$ слоев и $N_{K_{all}} = 1$ нейронов на выходном слое, $y_k^{(K_{all})}$ — выход нейронечеткой сети при подаче на ее вход вектора $\boldsymbol{x_k}$; в противном случае переход к шагу 4.

По сравнению с интерактивным обучением [39], в вышепредставленном алгоритме корректировка весов выполняется только после предъявления на вход сети всей обучающей выборки векторов, накопленной в течение целой эпохи.

Модель машины опорных векторов. Машина опорных векторов (МОВ) [6, 7] является одним из широко распространенных подходов, применяемых для решения задач классификации [124], регрессии [93] и прогнозирования [161]. Метод имеет простую геометрическую аналогию, которая связана с предположением, что элементы различных классов могут быть линейно разделены как принадлежащие различным подпространствам. Отсюда возникает идея о построении линейной гиперплоскости, которая обеспечивает необходимое разбиение классифицируемых точек. Очевидно, что такая гиперплоскость при условии ее существования не претендует на единственность, поэтому необходимо предоставить некоторый оптимизационный критерий, который позволил бы выделить одну наиболее подходящую поверхность среди всех подходящих на эту роль. Вполне естественным и разумным критерием для выбора разделяющей поверхности может служить условие максимизации зазора между ближайшими к этой гиперплоскости точками из разных классов.

На рисунке 2.13 изображены схематически элементы двух классов C_α и C_β. Эти элементы могут быть разделены несколькими различными плоскостями, описываемыми семейством уравнений вида $\boldsymbol{w}^T \cdot \boldsymbol{z} - b = 0$ и отличающимися друг от друга вектором нормали $\boldsymbol{w}$ (задающим наклон гиперплоскости) и параметром смещения b (задающим уровень подъема/спуска гиперплоскости). Обозначим левую часть данного уравнения через $y(\boldsymbol{z})$, т.е. $y(\boldsymbol{z}) = \boldsymbol{w}^T \cdot \boldsymbol{z} - b$. При подстановке в эту функцию конкретного значения вектора $\boldsymbol{z'}$ можно получить три различных варианта. Первый вариант $y(\boldsymbol{z'}) > 0$ соответствует тому случаю, когда вектор $\boldsymbol{z'}$ находится выше рассматриваемой гиперплоскости. Конкретно для изображенной на рисунке 2.13 гиперплоскости H получаем, что $y(\boldsymbol{z'}) = \boldsymbol{w}^T \cdot \boldsymbol{z'} - b > 0$ равносильно $\boldsymbol{z'} \in C_\alpha$. Во втором варианте неравенство $y(\boldsymbol{z'}) < 0$ означает, что вектор $\boldsymbol{z'}$ лежит ниже гиперплоскости, и поэтому $\boldsymbol{z'} \in C_\beta$. Наконец, третий вариант $y(\boldsymbol{z'}) = 0$ соответствует граничному случаю, когда вектор $\boldsymbol{z'}$ является решением данного уравнения, а значит, принадлежит разделяющей гиперплоскости.

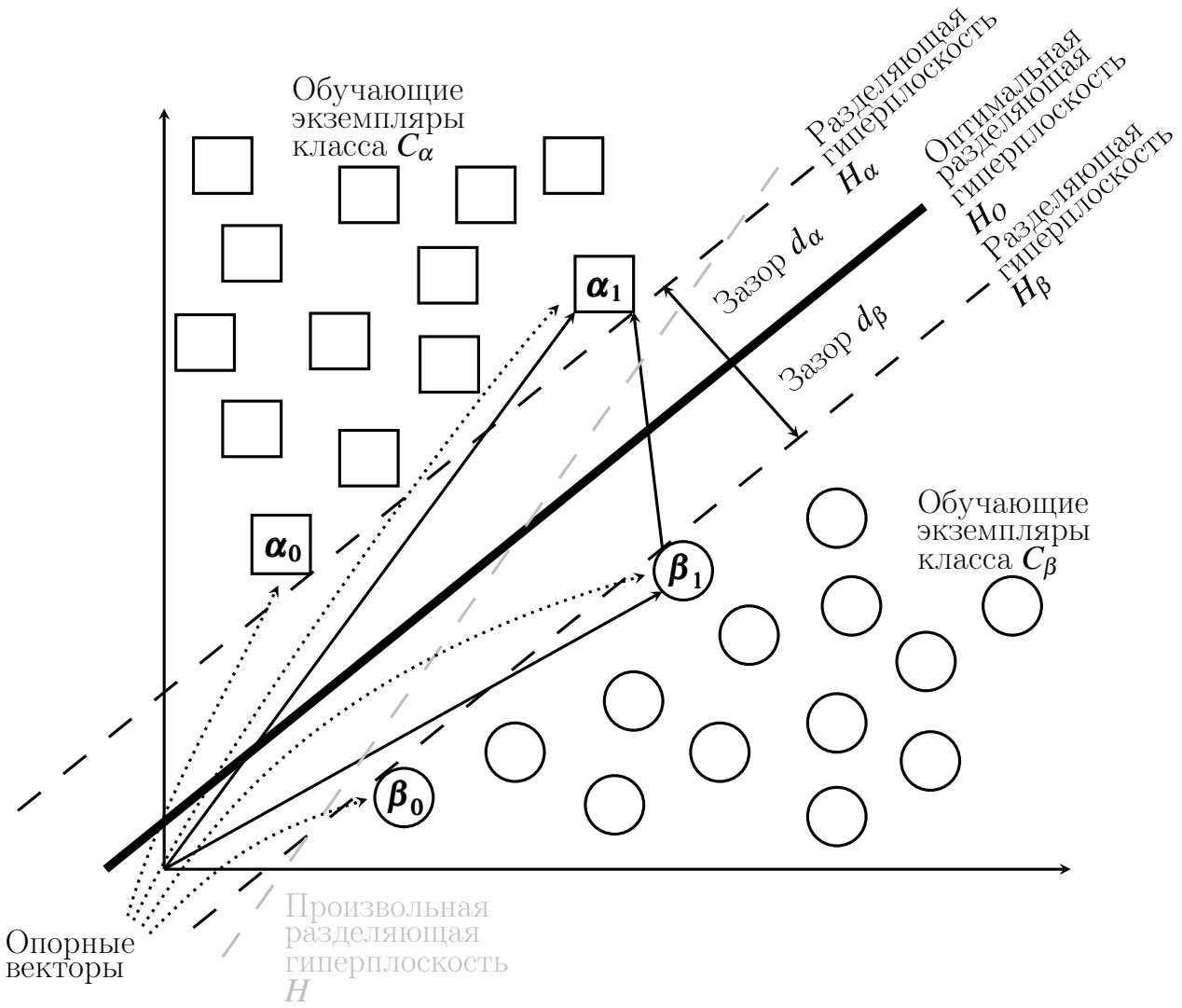


Рисунок 2.13 — Принцип построения оптимальной разделяющей гиперплоскости в машине опорных векторов

На рисунке 2.13 показано несколько разделяющих поверхностей. Две из них H_α и H_β, именуемые в дальнейшем граничными разделяющими гиперплоскостями и описываемые уравнениями $\boldsymbol{w}_{\boldsymbol{\alpha}}^T \cdot \boldsymbol{z} - b_\alpha = 0$ и $\boldsymbol{w}_{\boldsymbol{\beta}}^T \cdot \boldsymbol{z} - b_\beta = 0$, являются параллельными оптимальной гиперплоскости H_O, находятся на расстоянии d, равноудаленном от нее, и среди всех прочих разделяющих поверхностей, параллельных гиперплоскости H_O, являются наиболее близкими к элементам соответственно класса C_α и C_β. Ближайшие элементы, отмеченные на рисунке как $\boldsymbol{\alpha_0}$, $\boldsymbol{\alpha_1}$, $\boldsymbol{\beta_0}$, $\boldsymbol{\beta_1}$, называются опорными векторами, и именно эти векторы влияют на настраиваемые в процессе обучения параметры $\boldsymbol{w}$ и b. Поэтому отсюда можно утверждать, что алгоритм обучения „мягко реагирует“ на увеличение объема обучающей выборки.

Пусть оптимальная гиперплоскость H_O задается с помощью уравнения $\boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O = 0$, тогда $\boldsymbol{w}_{\boldsymbol{\alpha}} = \boldsymbol{w}_{\boldsymbol{O}}$, $b_\alpha = b_O + \varepsilon$, $\boldsymbol{w}_{\boldsymbol{\beta}} = \boldsymbol{w}_{\boldsymbol{O}}$, $b_\beta = b_O - \varepsilon$ в силу параллельности плоскостей ($\varepsilon > 0$). Не умаляя общности, можно считать, что $\varepsilon = 1$ (в противном случае этого можно добиться делением обеих частей уравнения на ε). После несложных преобразований уравнения двух разделяющих плоскостей H_α и H_β приобретают следующий вид: $\boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O = 1$ (верхняя гиперплоскость H_α) и $\boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O = -1$ (нижняя гиперплоскость H_β), а классы C_α и C_β представляются следующим образом: $C_\alpha = \left\{ \boldsymbol{z} \,\middle|\, \boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O \geqslant 1 \right\}$, $C_\beta = \left\{ \boldsymbol{z} \,\middle|\, \boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O \leqslant -1 \right\}$. Для нахождения величины d зазора (margin), представляющего собой расстояние между одним из опорных векторов и оптимальной разделяющей гиперплоскостью H_O, отметим, что зазор d равняется половинному расстоянию $d_\alpha + d_\beta$ между граничными плоскостями H_α и H_β в силу одного из требований к H_O. Из рисунка 2.13 видно, что значение $d_\alpha + d_\beta$ есть не что иное, как длина проекции разности какой-либо пары опорных векторов ($\boldsymbol{\alpha_1}$,$\boldsymbol{\beta_1}$) из разных классов на нормаль гиперплоскости $\boldsymbol{w}_{\boldsymbol{O}}$, а именно скалярное произведение единичной нормали $\frac{\boldsymbol{w}_{\boldsymbol{O}}}{\|\boldsymbol{w}_{\boldsymbol{O}}\|}$ оптимальной гиперплоскости H_O и разности векторов $\boldsymbol{\alpha_1} - \boldsymbol{\beta_1}$. Тем самым получаем следующее выражение: $2 \cdot d = \frac{\boldsymbol{w}_{\boldsymbol{O}}^T}{\|\boldsymbol{w}_{\boldsymbol{O}}\|} \cdot (\boldsymbol{\alpha_1} - \boldsymbol{\beta_1}) = \frac{\boldsymbol{w}_{\boldsymbol{O}}^T \cdot (\boldsymbol{\alpha_1} - \boldsymbol{\beta_1})}{\|\boldsymbol{w}_{\boldsymbol{O}}\|} = \frac{\boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{\alpha_1} - \boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{\beta_1}}{\|\boldsymbol{w}_{\boldsymbol{O}}\|} = \frac{b_O + 1 - (b_O - 1)}{\|\boldsymbol{w}_{\boldsymbol{O}}\|} = \frac{2}{\|\boldsymbol{w}_{\boldsymbol{O}}\|}$. Откуда $d = d_\alpha = d_\beta = \frac{1}{\|\boldsymbol{w}_{\boldsymbol{O}}\|}$. После полученных результатов можно описать функционирование модели машины опорных векторов следующим образом:

$$Y(\boldsymbol{z}) = \operatorname{sign}\left(\boldsymbol{w}_{\boldsymbol{O}}^T \cdot \boldsymbol{z} - b_O\right) = \operatorname{sign}\left(\sum_{i=1}^{n} w_{Oi} \cdot z_i - b_O\right),$$

где $\boldsymbol{w}_{\boldsymbol{O}} = (w_{O1}, \ldots, w_{On})^T$ и $\boldsymbol{z} = (z_1, \ldots, z_n)^T$. В отличие от нейронных сетей, в которых используется последовательная композиция операторов взвешенного суммирования и активационной функции, модель машины опорных векторов не требует столь сложных вычислений. Кроме того, ее алгоритм обучения существенно быстрее за счет явного решения оптимизационной задачи для нахождения оптимальной гиперплоскости вместо характерного для нейронных сетей итеративного процесса постепенной настройки весовых коэффициентов.

Рассмотрим алгоритм обучения машины опорных векторов при условии существования линейной гиперплоскости, корректно разделяющей все экземпляры обучающей выборки (алгоритм 11).

Алгоритм 11: Алгоритм обучения машины опорных векторов

---

1. Подготовка обучающих данных $\Upsilon_{\mathcal{X}^{(LS)}_{\{C_\alpha,C_\beta\}}} = \{(\boldsymbol{x_i},\bar{c}_i)\}_{i=1}^{M}$, где

$$\bar{c}_i = \frac{\alpha-\beta}{2}\cdot u_i + \frac{\alpha+\beta}{2} = \begin{cases} \beta, & \text{если } u_i = -1 \\ \alpha, & \text{если } u_i = 1 \end{cases} \text{ — возможные метки классов,}$$

$$u_i = [\boldsymbol{x_i} \in C_\alpha] - [\boldsymbol{x_i} \in C_\beta] = \begin{cases} -1, & \text{если } \boldsymbol{x_i} \in C_\beta \\ 1, & \text{если } \boldsymbol{x_i} \in C_\alpha \end{cases} \text{ — желаемый выход.}$$

2. Нахождение множителей Лагранжа $\lambda_i^{(O)}$ как результат решения оптимизационной задачи:

$$-\frac{1}{2}\cdot\sum_{i=1}^{M}\sum_{j=1}^{M}\lambda_i\cdot\lambda_j\cdot u_i\cdot u_j\cdot \boldsymbol{x_i}^T\cdot\boldsymbol{x_j} + \sum_{i=1}^{M}\lambda_i \to \max_{\lambda_1,\ldots,\lambda_M} \text{ при ограничениях}$$

$$\begin{cases} \sum\limits_{i=1}^{M}\lambda_i\cdot u_i \;\; = 0 \\ \qquad\quad \lambda_i \;\; \geqslant 0 \;\; \text{, где } i = 1,\ldots,M. \end{cases}$$

Для корректной классификации объектов из обучающей совокупности $\{\boldsymbol{x_i}\}_{i=1}^{M}$ необходимо и достаточно, чтобы значения выражений $\operatorname{sign}\left(\boldsymbol{w}^T\cdot\boldsymbol{x_i}-b\right)$ и u_i $(i=1,\ldots,M)$ одновременно были либо положительными, либо отрицательными, что равносильно выполнению неравенства $\operatorname{sign}\left(\boldsymbol{w}^T\cdot\boldsymbol{x_i}-b\right)\cdot u_i > 0$ $(i=1,\ldots,M)$. Принимая во внимание предположение о возможности линейного разделения обучающих экземпляров, отметим, что аргументы обоих множителей в его левой части по абсолютному значению не меньше единицы, поэтому $\left(\boldsymbol{w}^T\cdot\boldsymbol{x_i}-b\right)\cdot u_i \geqslant 1$ $(i=1,\ldots,M)$. Поскольку расстояние между оптимальной разделяющей гиперплоскостью и элементами, подлежащими классификации, является величиной, обратно пропорциональной норме вектора нормали к данной гиперплоскости, то максимизация этого зазора равносильна следующему опти-

мизационному критерию: $\Phi(\boldsymbol{w}) = \frac{1}{2} \cdot \|\boldsymbol{w}\|^2 = \frac{1}{2} \cdot \boldsymbol{w}^T \cdot \boldsymbol{w} \to \min\limits_{\boldsymbol{w}}$. По теореме Куна-Таккера [141] минимизация квадратичного функционала $\Phi(\boldsymbol{w})$ может быть решена при помощи метода множителей Лагранжа:

$$L\left(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M\right) = \frac{1}{2} \cdot \boldsymbol{w}^T \cdot \boldsymbol{w} - \sum_{i=1}^{M} \lambda_i \cdot \left(\left(\boldsymbol{w}^T \cdot \boldsymbol{x_i} - b\right) \cdot u_i - 1\right) \to \min_{\boldsymbol{w},b} \max_{\lambda_1,\ldots,\lambda_M}.$$

Необходимыми условиями для существования седловой точки $\left(\boldsymbol{w_O},b_O,\lambda_1^{(O)},\ldots,\lambda_M^{(O)}\right)$, т.е. точки, удовлетворяющей условию:

$$\max_{\lambda_1,\ldots,\lambda_M} L\left(\boldsymbol{w_O},b_O,\lambda_1,\ldots,\lambda_M\right) = L\left(\boldsymbol{w_O},b_O,\lambda_1^{(O)},\ldots,\lambda_M^{(O)}\right) = \\ = \min_{\boldsymbol{w},b} L\left(\boldsymbol{w},b,\lambda_1^{(O)},\ldots,\lambda_M^{(O)}\right),$$

являются условия зануления частных производных лагранжиана $L\left(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M\right)$ по переменным $\boldsymbol{w}$, b.

$$\begin{cases} \frac{\partial L(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M)}{\partial \boldsymbol{w}} &= \boldsymbol{0} \\ \frac{\partial L(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M)}{\partial b} &= 0 \end{cases} \Rightarrow \begin{cases} \boldsymbol{w} - \sum\limits_{i=1}^{M} \lambda_i \cdot u_i \cdot \boldsymbol{x_i} &= \boldsymbol{0} \\ \sum\limits_{i=1}^{M} \lambda_i \cdot u_i &= 0. \end{cases}$$

Лагранжиан приобретает следующий вид:

$$L\left(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M\right) = \frac{1}{2} \cdot \boldsymbol{w}^T \cdot \boldsymbol{w} - \sum_{i=1}^{M} \lambda_i \cdot u_i \cdot \boldsymbol{w}^T \cdot \boldsymbol{x_i} + b \cdot \sum_{i=1}^{M} \lambda_i \cdot u_i + \sum_{i=1}^{M} \lambda_i = \\ = \frac{1}{2} \cdot \sum_{i=1}^{M} \sum_{j=1}^{M} \lambda_i \cdot \lambda_j \cdot u_i \cdot u_j \cdot \boldsymbol{x_i}^T \cdot \boldsymbol{x_j} - \sum_{i=1}^{M} \sum_{j=1}^{M} \lambda_i \cdot \lambda_j \cdot u_i \cdot u_j \cdot \boldsymbol{x_i}^T \cdot \boldsymbol{x_j} + \sum_{i=1}^{M} \lambda_i = \\ = -\frac{1}{2} \cdot \sum_{i=1}^{M} \sum_{j=1}^{M} \lambda_i \cdot \lambda_j \cdot u_i \cdot u_j \cdot \boldsymbol{x_i}^T \cdot \boldsymbol{x_j} + \sum_{i=1}^{M} \lambda_i.$$

Искомые значения $\lambda_i^{(O)}$ $(i = 1,\ldots,M)$ вычисляются через решение задачи максимизации лагранжиана $L\left(\lambda_1,\ldots,\lambda_M\right) \equiv L\left(\boldsymbol{w},b,\lambda_1,\ldots,\lambda_M\right)$ при ограничениях $\sum_{i=1}^{M} \lambda_i \cdot u_i = 0$ и $\lambda_i \geqslant 0$ $(i = 1,\ldots,M)$.

3. Вычисление вектора нормали и свободного коэффициента в уравнении оптимальной разделяющей гиперплоскости.

Вектор $\boldsymbol{w_O}$ представляется следующим образом: $\boldsymbol{w_O}=\sum\limits_{i=1}^{M}\lambda_i^{(O)}\cdot u_i\cdot\boldsymbol{x_i}$. В самóй седловой точке $(\boldsymbol{w_O},b_O)$ согласно условию дополняющей нежесткости выполняется следующее тождество: $\lambda_i^{(O)}\cdot\left(\left(\boldsymbol{w_O^T}\cdot\boldsymbol{x_i}-b_O\right)\cdot u_i-1\right)=0$ для $i=1,\ldots,M$. Те векторы $\boldsymbol{x_{i_j}}$, где $i_j\in\left\{i_1,\ldots,i_{\widetilde{M}}\right\}\subseteq\{1,\ldots,M\}$, которым соответствуют ненулевые $\lambda_{i_j}^{(O)}$, являются опорными векторами и должны удовлетворять уравнению одной из граничных разделяющих гиперплоскостей H_α или H_β, откуда может быть вычислен коэффициент $b_O=\boldsymbol{w_O^T}\cdot\boldsymbol{x_{i_j}}-u_{i_j}$. Тем самым вектор нормали $\boldsymbol{w_O}$ представляется как линейная комбинация опорных векторов $\boldsymbol{x_{i_j}}$ $(j=1,\ldots,\widetilde{M})$.

4. Уточнение модели машины опорных векторов:

$$Y(\boldsymbol{z})=\operatorname{sign}\left(\sum_{j=1}^{\widetilde{M}}w_{i_j}\cdot\boldsymbol{x_{i_j}^T}\cdot\boldsymbol{z}-b_O\right),$$

где $w_{i_j}=\lambda_{i_j}^{(O)}\cdot u_{i_j}$ и оператор суммирования берется по индексному подмножеству $\left\{i_1,\ldots,i_{\widetilde{M}}\right\}$ обучающей выборки, которое соответствует только опорным векторам $\left\{\boldsymbol{x_{i_j}}\right\}_{j=1}^{\widetilde{M}}\subseteq\left\{\boldsymbol{x_i}\right\}_{i=1}^{M}$.

Когда объекты из разных классов не могут быть линейно разделены, используются два подхода, причем оба из них направлены на уменьшение значения функционала эмпирического риска $\Omega\left(Y,\mathcal{X}_{\mathcal{C}}^{(LS)}\right)$ на элементах обучающей выборки. Первый подход заключается в применении специальных нелинейных преобразований — ядер φ [39, 61, 187, 190], которые позволяют осуществлять переход к спрямляющим $\widehat{n}$-мерным пространствам. Предполагается, что в новом пространстве, полученном в результате отображения $\boldsymbol{\varphi}(\boldsymbol{z})=\left(\varphi_1(\boldsymbol{z}),\ldots,\varphi_{\widehat{n}}(\boldsymbol{z})\right)^T$, уже будет существовать гиперплоскость, удовлетворяющая ранее заданному критерию. Второй подход основан на введении штрафной функции, чтобы игнорировать некоторые из ложно классифицируемых объектов на основе минимизации или их общего количества, или их суммарного расстояния до разделяющей гиперплоскости. В первом случае осуществляется поиск такой разделяющей гиперплоскости, которая доставляет минимальное значение следующей характеристической функции $\sum_{i=1}^{M}\left[Y(\boldsymbol{x_i})\neq u_i\right]$. Во втором случае в роли целевой функ-

ции, также подлежащей минимизации, выступает $\sum_{i=1}^{M}[Y(\boldsymbol{x_i}) \neq u_i]\cdot\mathrm{dist}(\boldsymbol{x_i},H_O)$, где $\mathrm{dist}(\cdot,\cdot)$ — функция расстояния между указанной парой аргументов (вектор, плоскость) в рамках заданной метрики.

Алгоритм классификации сетевых соединений при помощи одного из рассмотренных бинарных классификаторов представлен в алгоритме 12.

Алгоритм 12: Классификация сетевых соединений при помощи $\widetilde{Y}$

---

1. Выбор бинарного классификатора $\widetilde{Y}$ и задание его параметров.
2. Выбор множества классов $\widetilde{\mathcal{C}}_{\widetilde{ij}} = \widetilde{\mathcal{C}}_{\widetilde{i}} \bigvee \widetilde{\mathcal{C}}_{\widetilde{j}} \subseteq \mathcal{C}$ для обучения детектора $\widetilde{Y}$.
3. Подготовка обучающих данных $\Upsilon_{\mathcal{X}^{(LS)}_{\widetilde{\mathcal{C}}_{\widetilde{ij}}}} = \left\{\left(\boldsymbol{x}_{\widetilde{\boldsymbol{k}}},\bar{c}_{\widetilde{k}}\right)\right\}_{\widetilde{k}=1}^{\widetilde{M}}$, где

   $\widetilde{M} = \#\mathcal{X}^{(LS)}_{\widetilde{\mathcal{C}}_{\widetilde{ij}}}$ — мощность обучающего множества для детектора $\widetilde{Y}$,

   $$\bar{c}_{\widetilde{k}} = \begin{cases} \widetilde{i}, & \text{если } u_{\widetilde{k}} = -1 \vee u_{\widetilde{k}} = 0 \\ \widetilde{j}, & \text{если } u_{\widetilde{k}} = 1 \end{cases} \text{ — возможные метки классов,}$$

   $$u_{\widetilde{k}} = \begin{cases} \bar{y}_{\widetilde{i}} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, & \text{если } \exists\widetilde{C} \in \widetilde{\mathcal{C}}_{\widetilde{i}}\ \boldsymbol{x}_{\widetilde{\boldsymbol{k}}} \in \widetilde{C} \\ \bar{y}_{\widetilde{j}} = 1, & \text{если } \exists\widetilde{C} \in \widetilde{\mathcal{C}}_{\widetilde{j}}\ \boldsymbol{x}_{\widetilde{\boldsymbol{k}}} \in \widetilde{C} \end{cases} \text{ — ожидаемый выход.}$$

   Значение параметра $\bar{y}_{\widetilde{i}}$ выбирается равным -1 или 0 в зависимости от типа бинарного классификатора $\widetilde{Y}$ и функции активации/принадлежности на его выходном слое.
4. Обучение бинарного классификатора $\widetilde{Y}$ на данных $\Upsilon_{\mathcal{X}^{(LS)}_{\widetilde{\mathcal{C}}_{\widetilde{ij}}}}$.
5. Классификация входного объекта $\boldsymbol{z}$, представляющего собой вектор признаков анализируемого сетевого соединения: если $\widetilde{Y}(\boldsymbol{z}) < \bar{h}_{\widetilde{ij}}$, то объект $\boldsymbol{z}$ относится к одному из классов $\widetilde{\mathcal{C}}_{\widetilde{i}}$; если $\widetilde{Y}(\boldsymbol{z}) \geqslant \bar{h}_{\widetilde{ij}}$, то объект $\boldsymbol{z}$ относится к одному из классов $\widetilde{\mathcal{C}}_{\widetilde{j}}$. Здесь $\bar{h}_{\widetilde{ij}} = \frac{\bar{y}_{\widetilde{i}}+\bar{y}_{\widetilde{j}}}{2}$ может быть интерпретирован как порог активации бинарного классификатора $\widetilde{Y}$. В частности, если $\widetilde{\mathcal{C}}_{\widetilde{i}} = \{C_0\} \wedge \widetilde{Y}(\boldsymbol{z}) < \bar{h}_{\widetilde{ij}}$, то сетевое соединение, представленное вектором признаков $\boldsymbol{z}$, распознается как нормальное.

2.5 Методика иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений

Общее представление предлагаемой методики для обнаружения аномальных сетевых соединений показано на рисунке 2.14 и состоит из следующих пяти этапов:

1. построение дерева классификаторов;
2. формирование параметров сетевых соединений;
3. предобработка параметров сетевых соединений;
4. иерархический обход дерева классификаторов в ширину;
5. обнаружение аномальных сетевых соединений.

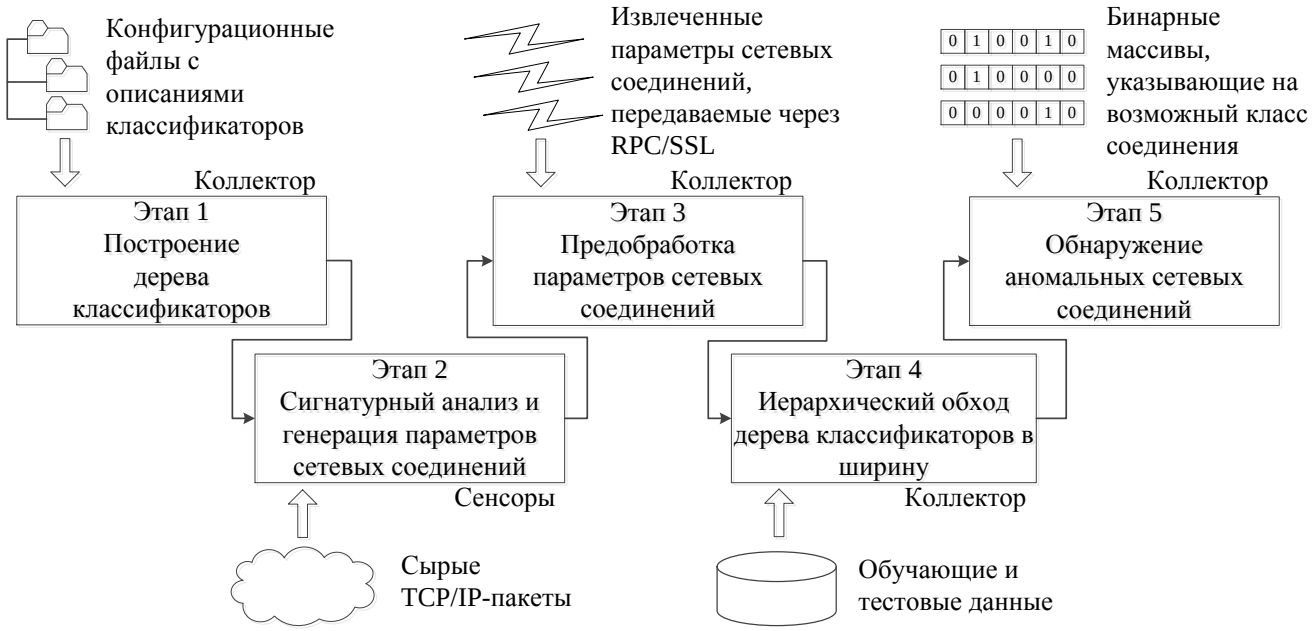


Рисунок 2.14 — Основные этапы методики иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений (без детализации)

Более подробное представление данной методики с указанием компонентов СОА и действий, выполняемых оператором и системой, показано на рисунке 2.15.

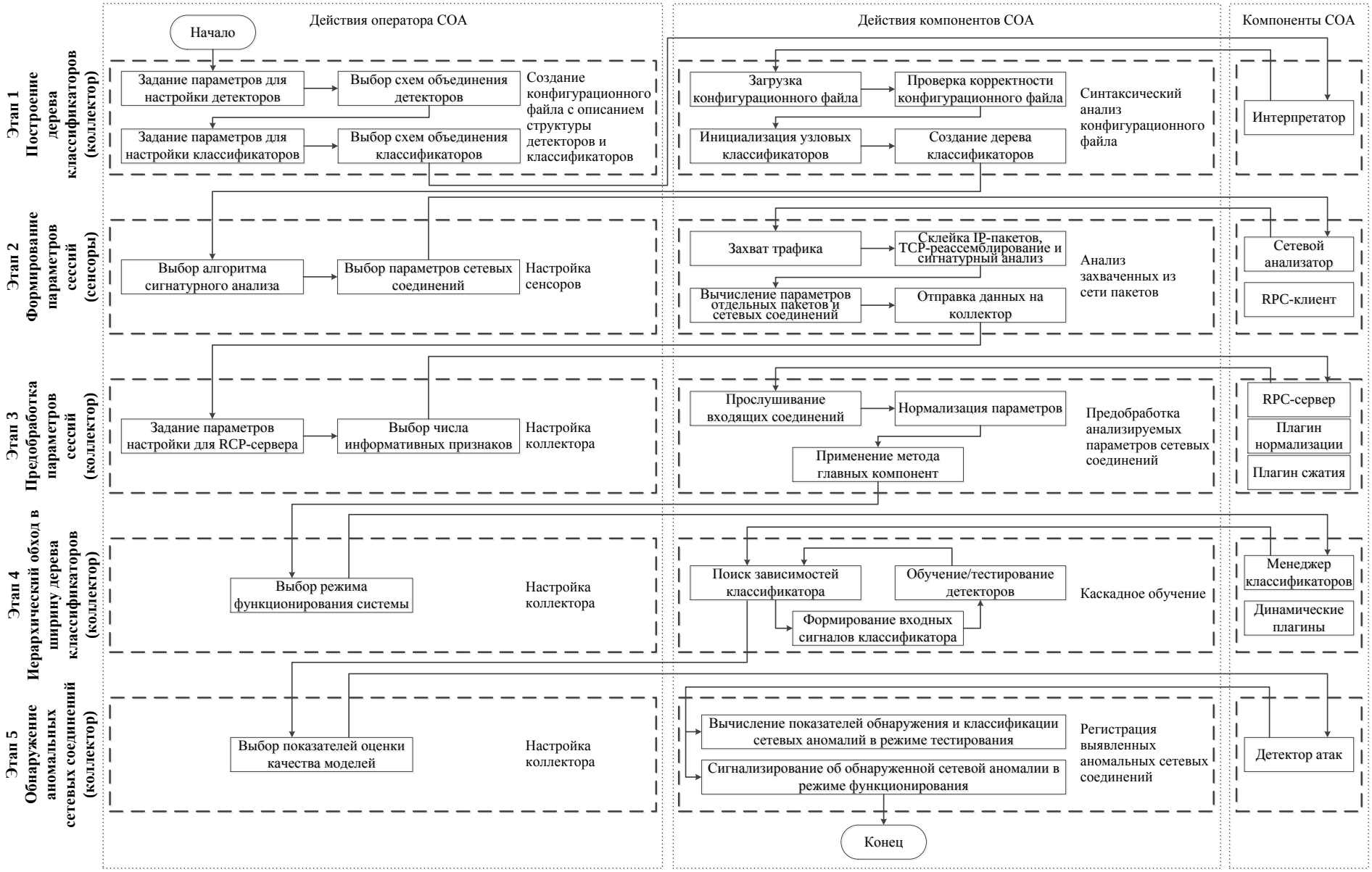


Рисунок 2.15 — Основные этапы методики иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений (с детализацией)

Первый этап методики может быть охарактеризован как подготовительный, он включает в себя выбор структуры отдельных бинарных классификаторов (детекторов): размерности и числа слоев, параметров и алгоритмов обучения, типов функций активации, функций принадлежности и ядерных функций. Для каждого детектора может быть составлен набор обучающих правил. Задавая различную совокупность таких наборов правил, можно сформировать группу детекторов, каждый из которых построен на основе одной из приведенных в разделе 2.4 моделей. Детекторы внутри каждой такой группы объединяются в классификатор на основе подходов один-ко-всем (one-vs-all), один-к-одному (one-vs-one) или их различных производных вариаций [106, 148]. В первом подходе каждый детектор $F_{jk}^{(i)} : \mathbb{R}^n \to \{0,1\}$ $(k = 1,\ldots,m)$ обучается на данных $\{(\boldsymbol{x_l}, [\bar{c}_l = k])\}_{l=1}^{M}$[4], и функционирование группы детекторов $F_j^{(i)}$ описывается при помощи исключающего принципа:

$$F_j^{(i)}(\boldsymbol{z}) = \begin{cases} \{0\}, & \text{если } \forall k \in \{1,\ldots,m\} \;\; F_{jk}^{(i)}(\boldsymbol{z}) = 0 \\ \left\{k \,\middle|\, F_{jk}^{(i)}(\boldsymbol{z}) = 1\right\}_{k=1}^{m}, & \text{иначе.} \end{cases}$$

Во втором подходе каждый из $C_{m+1}^2 = \frac{(m+1)\cdot m}{2}$ детекторов $F_{jk_0k_1}^{(i)}$ обучается на множестве объектов, принадлежащих только двум классам с метками k_0 и k_1, – $\{(\boldsymbol{x}_l, 0 \,|\bar{c}_l = k_0)\}_{l=1}^{M} \bigcup \{(\boldsymbol{x}_l, 1 \,|\bar{c}_l = k_1)\}_{l=1}^{M}$, где $0 \leqslant k_0 < k_1 \leqslant m$, и функционирование группы детекторов $F_j^{(i)}$ задается с помощью голосования max-wins:

$$F_j^{(i)} = \left\{ \arg \max_{\bar{c} \in \{0,\ldots,m\}} \sum_{k=\bar{c}+1}^{m} \left[F_{j\bar{c}k}^{(i)}(\boldsymbol{z}) = 0 \right] + \sum_{k=0}^{\bar{c}-1} \left[F_{jk\bar{c}}^{(i)}(\boldsymbol{z}) = 1 \right] \right\}.$$

В качестве одной из производных вариаций предыдущих подходов для комбинирования детекторов может быть упомянуто классификационное бинарное дерево [2]. Формально такая структура задается рекурсивно следующим образом:

$$CBT_\mu = \begin{cases} \langle F_{jL_\mu R_\mu}^{(i)}, CBT_{L_\mu}, CBT_{R_\mu} \rangle, & \text{если } \#\mu \geqslant 2 \\ \mu, & \text{если } \#\mu = 1. \end{cases}$$

[4] В этом разделе под вторым элементом, указанным в наборе обучающих данных, понимается желаемый выход u_l детектора. Сам детектор представляется как композиция пороговой функции и функции Y: $\left[Y(\boldsymbol{x_l}) \geqslant \bar{h}_{\widetilde{ij}}\right]$.

Здесь $\mu = \{0,\ldots,m\}$ — исходный набор меток классов, $L_\mu \subsetneq \mu$ — произвольно сгенерированное или предопределенное пользователем подмножество μ ($\#L_\mu < \#\mu$), $R_\mu = \mu \setminus L_\mu$, CBT_{L_μ} — левое классификационное поддерево, CBT_{R_μ} — правое классификационное поддерево, $F^{(i)}_{jL_\mu R_\mu}$ — узловой детектор, обученный на элементах множества $\{(\boldsymbol{x_l},0)\,|\bar{c}_l \in L_\mu\}_{l=1}^{M} \bigcup \{(\boldsymbol{x_l},1)\,|\bar{c}_l \in R_\mu\}_{l=1}^{M}$, т.е. выходной результат детектора настраивается таким образом, чтобы он равнялся 0, если входной объект $\boldsymbol{x_l}$ принадлежит классу с меткой $\bar{c}_l \in L_\mu$, и 1, если объект $\boldsymbol{x_l}$ принадлежит классу с меткой $\bar{c}_l \in R_\mu$. Поэтому функционирование группы детекторов $F^{(i)}_j$, представленных в виде узлов такого дерева, описывается с помощью рекурсивной функции $\phi^{(i)}_j$, задающей последовательную дихотомию множества μ:

$$F^{(i)}_j = \phi^{(i)}_j(\mu,\boldsymbol{z}),$$

$$\phi^{(i)}_j(\mu,\boldsymbol{z}) = \begin{cases} \mu, & \text{если } \#\mu = 1 \\ \phi^{(i)}_j(L_\mu,\boldsymbol{z}), & \text{если } \#\mu \geqslant 2 \wedge F^{(i)}_{jL_\mu R_\mu}(\boldsymbol{z}) = 0 \\ \phi^{(i)}_j(R_\mu,\boldsymbol{z}), & \text{если } \#\mu \geqslant 2 \wedge F^{(i)}_{jL_\mu R_\mu}(\boldsymbol{z}) = 1. \end{cases}$$

Применение функции $\phi^{(i)}_j$ к исходному набору меток классов и классифицируемому объекту позволяет осуществлять однозначный поиск метки класса этого объекта. Это объясняется тем, что поскольку по мере спуска вниз по классификационному дереву происходит дизъюнктное разбиение множества меток классов, то после достижения и срабатывания терминального детектора остается только одна возможная метка для классификации входного объекта $\boldsymbol{z}$ в качестве выходного результата $F^{(i)}_j$. Поэтому для классификационного дерева невозможны конфликтные случаи при классификации объектов, которые могут иметь место для двух других подходов комбинирования.

Другим подходом является направленный ациклический граф, который организует $C^2_{m+1} = \frac{(m+1)\cdot m}{2}$ детекторов в связную динамическую структуру, которая может быть задана следующей формулой:

$$DAG_\mu = \begin{cases} \langle F^{(i)}_{j\mu k_0 k_1}, DAG_{\mu\setminus\{k_0\}}, DAG_{\mu\setminus\{k_1\}}\rangle, & \text{если } \#\mu \geqslant 2, \text{ где } k_0 \in \mu, k_1 \in \mu \\ \mu, & \text{если } \#\mu = 1. \end{cases}$$

Здесь, как и в подходе один-к-одному, каждый узловой детектор $F^{(i)}_{j\mu k_0 k_1}$ обучается на элементах $\{(\boldsymbol{x}_l, 0 \,|\bar{c}_l = k_0)\}_{l=1}^{M} \bigcup \{(\boldsymbol{x}_l, 1 \,|\bar{c}_l = k_1)\}_{l=1}^{M}$ ($k_0 < k_1$). Обход рассматриваемого графа выполняется при помощи рекурсивной функции $\xi^{(i)}_j$, задающей поэлементное „отщепление“ от множества μ:

$$F^{(i)}_j = \xi^{(i)}_j(\mu, \boldsymbol{z}),$$

$$\xi^{(i)}_j(\mu, \boldsymbol{z}) = \begin{cases} \mu, & \text{если } \#\mu = 1 \\ \xi^{(i)}_j(\mu \setminus \{k_1\}, \boldsymbol{z}), & \text{если } \#\mu \geqslant 2 \wedge F^{(i)}_{j\mu k_0 k_1}(\boldsymbol{z}) = 0 \\ \xi^{(i)}_j(\mu \setminus \{k_0\}, \boldsymbol{z}), & \text{если } \#\mu \geqslant 2 \wedge F^{(i)}_{j\mu k_0 k_1}(\boldsymbol{z}) = 1. \end{cases}$$

Если детектор $F^{(i)}_{j\mu k_0 k_1}$ голосует за k_0-ый класс для объекта $\boldsymbol{z}$, т.е. $F^{(i)}_{j\mu k_0 k_1}(\boldsymbol{z}) = 0$, то из множества μ удаляется метка k_1 как заведомо неверная, в противном случае исключается метка k_0. Процесс повторяется до тех пор, пока множество μ не вырождается в одноэлементное. В таблице 5 приведены характеристики рассмотренных схем объединения детекторов в многоклассовую модель, предназначенную для соотнесения входного объекта одной или нескольким из $(m+1)$ меток классов. На рисунках 2.16, 2.17 изображены примеры классификационного бинарного дерева и направленного ациклического графа.

Таблица 5 — Характеристики схем объединения детекторов

Схема объединения	Число детекторов, подлежащих обучению	Минимальное число детекторов, задействованных при классификации объектов	Максимальное число детекторов, задействованных при классификации объектов
Один-ко-всем	m	m	m
Один-к-одному	$\frac{(m+1)\cdot m}{2}$	$\frac{(m+1)\cdot m}{2}$	$\frac{(m+1)\cdot m}{2}$
Классификационное бинарное дерево	m	1	m
Направленный ациклический граф	$\frac{(m+1)\cdot m}{2}$	m	m

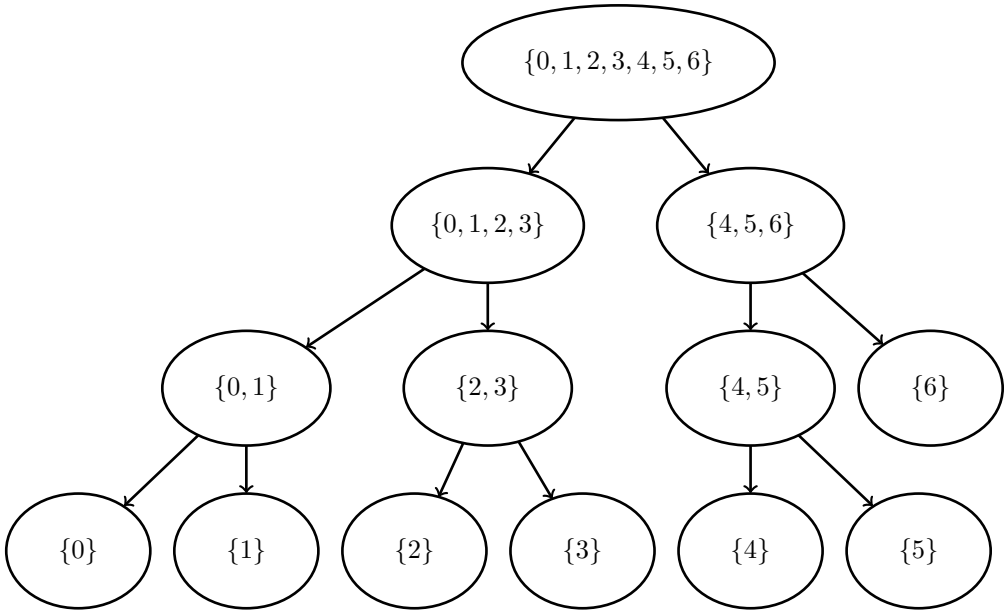


Рисунок 2.16 — Пример классификационного бинарного дерева

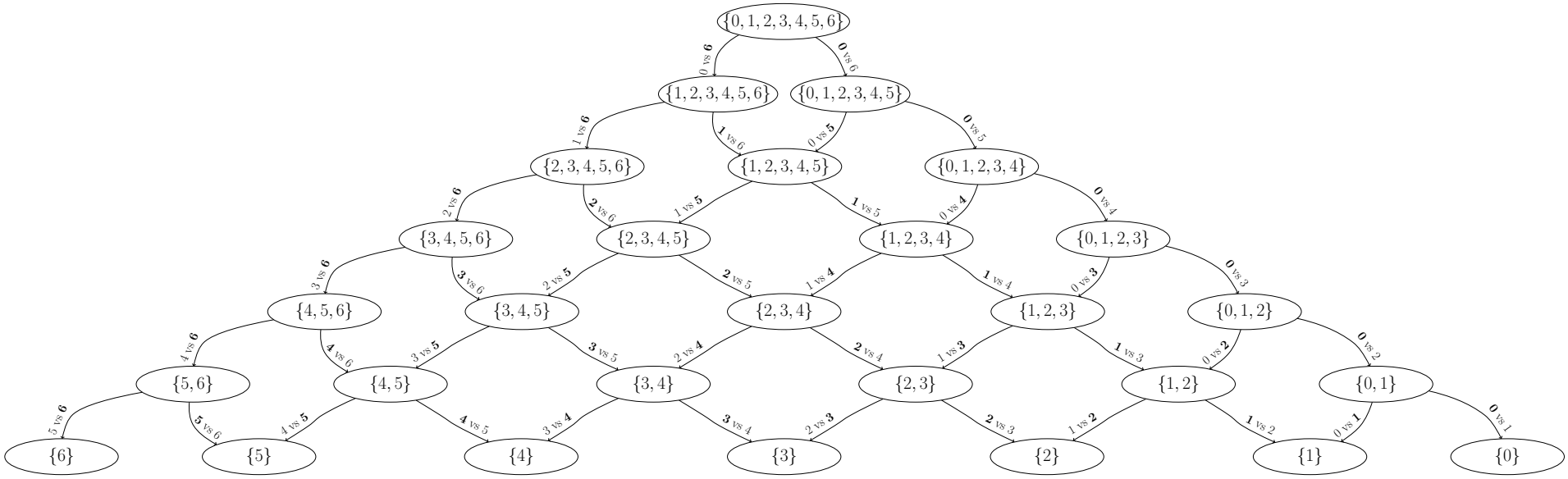


Рисунок 2.17 — Пример направленного ациклического графа

Из рассмотренных четырех схем только одна, а именно классификационное бинарное дерево, обладает переменным числом детекторов, которые могут использоваться в процессе классификации объектов. Минимальное значение достигается, когда активируется детектор $F^{(i)}_{jL_\mu R_\mu}$, расположенный в корне дерева и обученный для распознавания только одного класса объектов среди всех остальных, и $F^{(i)}_{jL_\mu R_\mu}(\boldsymbol{z}) = 0$ ($F^{(i)}_{jL_\mu R_\mu}(\boldsymbol{z}) = 1$) т.е. когда $\#L_\mu = 1$ ($\#R_\mu = 1$). Максимальное значение достигается, когда дерево представляется последовательным списком и активируется наиболее удаленный в нем детектор. В случае сбалансированного дерева этот показатель может составлять величину $\lfloor \log_2(m+1) \rfloor$ или $\lceil \log_2(m+1) \rceil$. На рисунке 2.18 представлен пример, когда каждый классификатор $F^{(i)}$ ($i = 1, \ldots, P$) содержит q_i групп $F^{(i)}_j$ ($j = 1, \ldots, q_i$), каждая из которых объединяет m детекторов $F^{(i)}_{jk}$ ($k = 1, \ldots, m$) при помощи подхода один-ко-всем. Каждая из групп детекторов $F^{(i)}_j$ обучается на различных случайных бутстреп-подвыборках, которые могут включать повторяющиеся и переупорядоченные элементы из исходного обучающего набора $\Upsilon_{\mathcal{X}^{(LS)}_C}$. Объединение групп $F^{(i)}_j$ в классификатор $F^{(i)}$ осуществляется на основе гибридного правила, представляющего собой смесь голосования большинством и голосования max-wins:

$$F^{(i)}(\boldsymbol{z}) = \left\{ \bar{c} \left| \underbrace{\sum_{j=1}^{q_i} \left[\bar{c} \in F^{(i)}_j(\boldsymbol{z}) \right]}_{\Xi_i(\bar{c})} > \frac{1}{2} \cdot q_i \wedge \Xi_i(\bar{c}) = \max_{\bar{c}' \in \{0,\ldots,m\}} \Xi_i(\bar{c}') \right. \right\}_{\bar{c}=0}^{m}.$$

Отметим, что в данной формуле за счет требования $\Xi_i(\bar{c}) > \frac{1}{2} \cdot q_i$ классификатор $F^{(i)}$ становится неспособным разрешать конфликты, которые возникают, к примеру, при условии $\# \left\{ \bar{c} \left| \Xi_i(\bar{c}) = \frac{1}{2} \cdot q_i \wedge \Xi_i(\bar{c}) = \max_{\bar{c}' \in \{0,\ldots,m\}} \Xi_i(\bar{c}') \right. \right\}_{\bar{c}=0}^{m} = 2$ (в этом случае выходом классификатора является пустое множество $\varnothing$). Для устранения этого недостатка автор настоящего диссертационного исследования разработал механизм разрешения таких конфликтных случаев, схожий с описанным в разделе 2.2 и основанный на непосредственных выходах детекторов, входящих в состав групп классификатора $F^{(i)}$.

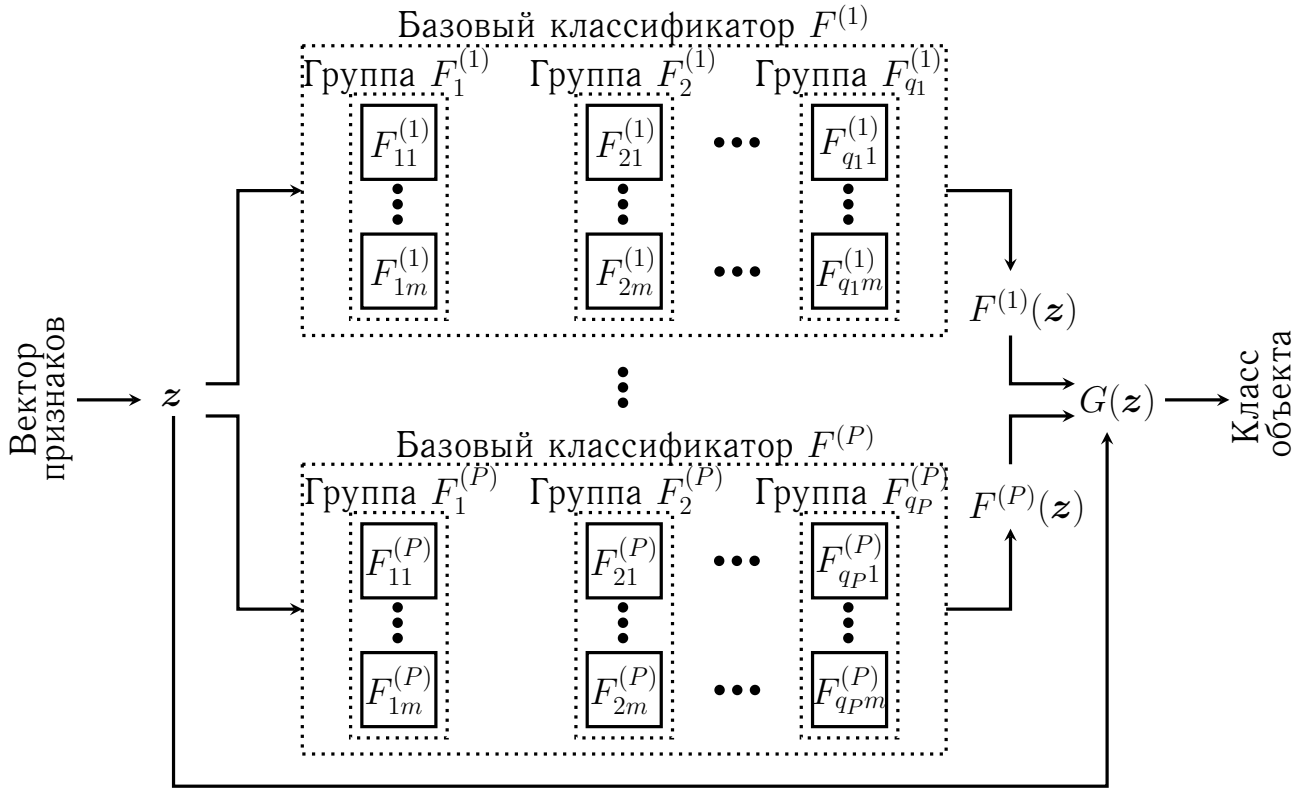


Рисунок 2.18 — Пример схемы объединения детекторов в многоклассовую модель

Для построения коллективного правила (агрегирующей композиции) G [10, 144, 212], объединяющего выходные результаты классификаторов $F^{(i)}$, были реализованы следующие подходы:

1. метод мажоритарного голосования (ММГ), или метод голосования большинством (формула 1.1);
2. метод взвешенного голосования (МВГ), приписывающий классификаторам весовые коэффициенты (формула 1.2);
3. метод многоярусной укладки (ММУ), дополненный введением атрибута — номера кластера по методу K-средних (формула 1.3);
4. метод Фикса—Ходжеса (МФХ), представляющий собой объединение классификаторов с использованием арбитра на основе динамических областей компетентности и метода ближайших соседей (формула 1.4).

Для выполнения задачи синтаксического анализа был реализован интерпретатор, поддерживающий операции условного ветвления, конкатенации векторов, векторного суммирования, покомпонентного произведения и деления.

В процессе работы интерпретатора проверяется корректность обрабатываемого конфигурационного файла, и инициализируются поля объектов внутри строящегося дерева классификаторов. За счет использования такой структуры в рамках предложенной методики становится возможным строить многоуровневые схемы, в которых блоки, отмеченные как $F^{(1)}, \ldots, F^{(P)}$, будут содержать схожий с рисунком 2.18 фрагмент. Описание этого программного средства, а также пример содержимого конфигурационного файла, обрабатываемого им, представлены в разделе 3.2.

Данная методика подразумевает распределенную архитектуру реализующих ее систем, в которых сбор данных осуществляется вторичными узлами — сенсорами, а вся обработка агрегированных потоков данных выполняется на централизованном сервере — коллекторе. Второй этап методики, выполняемый на стороне сенсоров, заключается в применении разработанного алгоритма сборки сырых пакетов в сетевые соединения, выделении их параметров и выполнении сигнатурного анализа с использованием нескольких разработанных параллельных модификаций алгоритмов шаблонного поиска подстроки. С этой целью было исследовано быстродействие алгоритмов Ахо—Корасик и Бойера—Мура на выбранных сигнатурных записях Snort, и реализованы их улучшенные аналоги при помощи технологий OpenMP и CUDA. Экспериментальное сравнение этих алгоритмов представлено в разделе 3.1. Был реализован событийно-ориентированный анализатор сетевого трафика, с помощью которого было извлечено 106 сетевых параметров, среди которых можно назвать продолжительность соединения, используемую сетевую службу, интенсивность отправки хостом специальных пакетов, число активных соединений между конкретной парой IP-адресов (один из критериев DoS-атак), признак изменения масштабирования TCP-окна после фактического установления сессии, текущее состояние TCP-соединения, различные признаки наличия сканирующих пакетов на уровнях TCP, UDP, ICMP и IP (15 различных кодов сканирования) и пр. Классификация этих параметров показана на рисунке 2.19. Описание внутреннего устройства сетевого анализатора вместе с поддерживаемыми в нем функциями представлено в разделе 3.2.

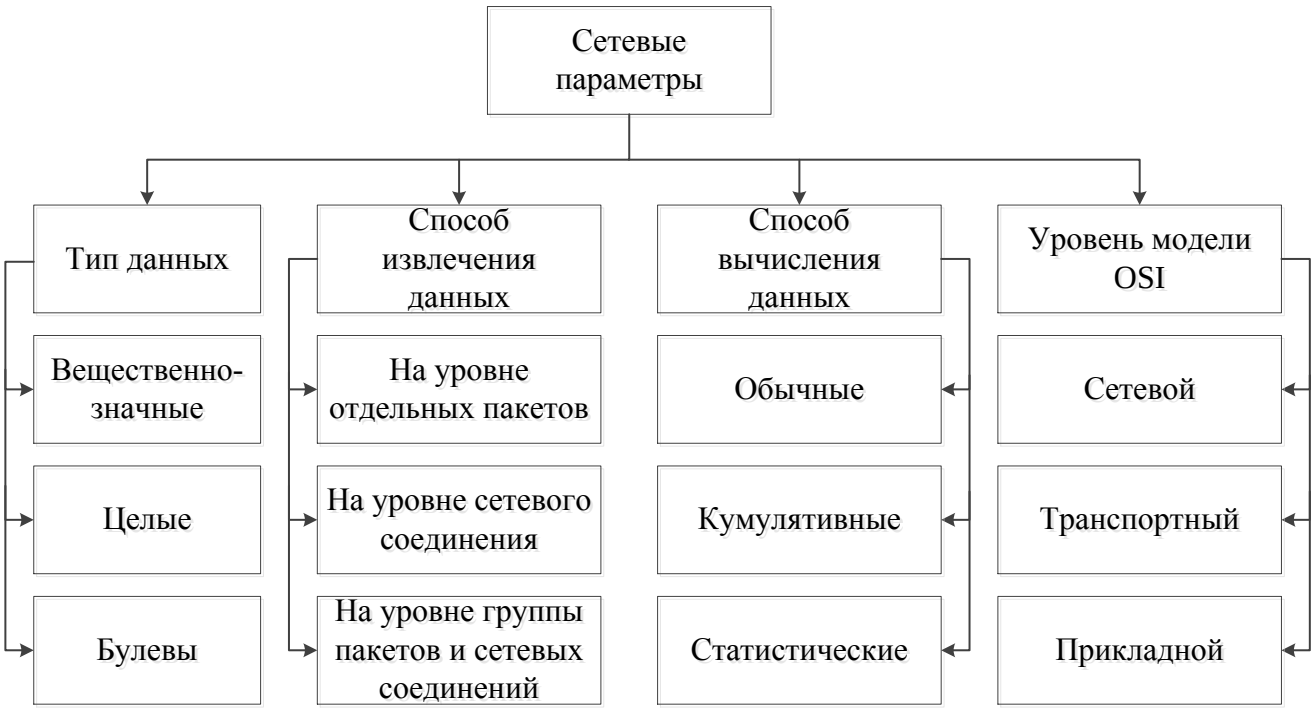


Рисунок 2.19 — Классификация сетевых параметров

Для измерения величины интенсивности отправки/приема пакетов использовался адаптированный метод скользящей средней. Суть метода заключается в разбиении заданного временного интервала $\Delta_0^{(L)} = [0,L]$ длиной L, в течение которого производится непрерывное наблюдение за рядом параметров, на несколько более мелких интервалов $\Delta_0^{(L')}, \Delta_\delta^{(L')}, \ldots, \Delta_{\delta\cdot(k-1)}^{(L')}$ одинаковой длины $0 < L' \leqslant L$, начало каждого из которых имеет смещение $0 < \delta \leqslant L'$ относительно начала предыдущего интервала (рисунок 2.20). Причем $\bigcup_{i=0}^{k-1} \Delta_{\delta\cdot i}^{L'} \subseteq \Delta_0^{(L)}$ и $\bigcup_{i=0}^{k} \Delta_{\delta\cdot i}^{L'} \supsetneq \Delta_0^{(L)}$, поэтому $k = 1 + \lfloor \frac{L-L'}{\delta} \rfloor$. В течение промежутков времени $\Delta_0^{(L')}, \ldots, \Delta_{\delta\cdot(k-1)}^{(L')}$ делаются слепки значений $\omega_0, \ldots \omega_{k-1}$ параметров, и их средняя величина (интенсивность) $\bar{\omega}$ в рамках временного окна длины L' рассчитывается по формуле $\bar{\omega} = \frac{1}{k} \cdot \sum_{i=0}^{k-1} \omega_i$. В настоящей диссертационной работе использовался интервал со значением параметра L, равным пяти секундам. Длина сглаживающего интервала L' была выбрана равной одной секунде. Смещение δ было установлено в полсекунды. Предполагается, что подобный подход позволяет устранить редкие по частоте и случайные сетевые всплески и тем самым снизить число ложных срабатываний.

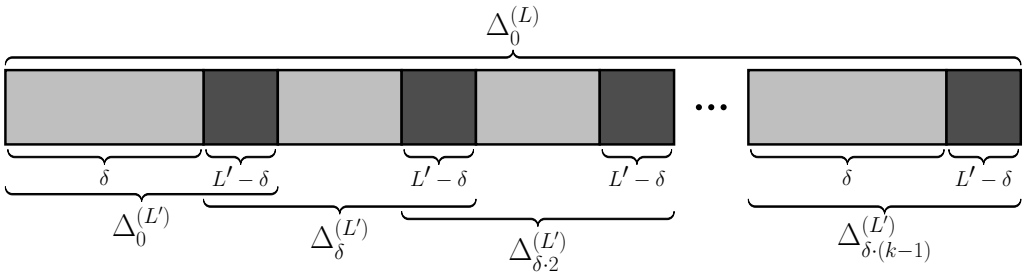


Рисунок 2.20 — Иллюстрация метода скользящей средней

Третий этап методики начинается с прослушивания входящих от сенсоров пакетов, передаваемых по протоколу RPC/SSL и содержащих вычисленные параметры соединений. Для обеспечения взаимодействия коллектора и сенсоров выбор пал в сторону именно такой связки протоколов, поскольку они гарантируют и быструю, и безопасную отправку данных. RPC является хорошо зарекомендовавшей себя и успешно прошедшей испытание временем технологией, которая позволяет без труда организовать компактную передачу бинарных потоков данных. SSL, в свою очередь, широко используется для создания шифрованного канала между передатчиками данных.

Перед непосредственным обучением детекторов выполняется предобработка данных параметров для уменьшения эффекта их сильной изменчивости. Многие методы, включая нейронные сети и метод главных компонент, чувствительны к такого рода флуктуациям и требуют, чтобы все признаки обрабатываемых векторов имели одинаковый масштаб. Поэтому первый шаг предобработки каждого компонента x_{ij} вектора $\boldsymbol{x_i} \in \{\boldsymbol{x_k}\}_{k=1}^{M}$ включает его нормализацию при помощи функции $f(x_{ij}) = \frac{x_{ij}-x_j^{(min)}}{x_j^{(max)}-x_j^{(min)}}$ (в случае $x_j^{(max)} = x_j^{(min)}$ можно полагать $f(x_{ij}) = 0$), где $x_j^{(min)} = \min\limits_{1\leqslant i\leqslant M} x_{ij}$ и $x_j^{(max)} = \max\limits_{1\leqslant i\leqslant M} x_{ij}$. Второй шаг нормализации — это уменьшение числа значимых признаков, которое достигается при помощи метода главных компонент [129, 171], описываемого как последовательность следующих шагов:

1. Вычисление математического ожидания случайного вектора, представленного в данном случае в виде элементов набора обучающих данных $\left\{\boldsymbol{x_i} = \{x_{ij}\}_{j=1}^{n}\right\}_{i=1}^{M}$:

$$\bar{\boldsymbol{x}} = (\bar{x}_1,\ldots,\bar{x}_n)^T = E\left[\{\boldsymbol{x_i}\}_{i=1}^{M}\right] = \frac{1}{M}\cdot\sum_{i=1}^{M}\boldsymbol{x_i} = \left(\frac{1}{M}\cdot\sum_{i=1}^{M}x_{i1},\ldots,\frac{1}{M}\cdot\sum_{i=1}^{M}x_{in}\right)^T.$$

2. Формирование элементов несмещенной теоретической ковариационной матрицы $\Sigma = (\sigma_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$:

$$\sigma_{ij} = \frac{1}{M-1} \cdot \sum_{k=1}^{M} (x_{ki} - \bar{x}_i) \cdot (x_{kj} - \bar{x}_j).$$

3. Нахождение собственных чисел $\{\lambda_i\}_{i=1}^{n}$ и собственных векторов $\{\boldsymbol{\nu_i}\}_{i=1}^{n}$ матрицы Σ как корней уравнений (с этой целью использовался метод вращений Якоби):

$$\begin{cases} \det(\Sigma - \lambda \cdot \boldsymbol{I}) &= 0 \\ (\Sigma - \lambda \cdot I) \cdot \boldsymbol{\nu} &= \mathbf{0}, \end{cases}$$

где $\boldsymbol{I}$ — единичная матрица размером $n \times n$.

4. Ранжирование собственных чисел $\{\lambda_i\}_{i=1}^{n}$ в порядке их убывания и соответствующих им собственных векторов $\{\boldsymbol{\nu_i}\}_{i=1}^{n}$:

$$\lambda_1 \geqslant \lambda_2 \geqslant \ldots \lambda_n \geqslant 0.$$

5. Отбор необходимого числа $\widehat{n} \leqslant n$ главных компонент:

$$\widehat{n} = \min \{j | \varsigma(j) \geqslant \varepsilon\}_{j=1}^{n},$$

где $\varsigma(j) = \frac{\sum_{i=1}^{j} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$ — мера информативности [1], $0 \leqslant \varepsilon \leqslant 1$ — экспертно выбираемая величина.

6. Центрирование вектора признаков $\boldsymbol{z}$: $\boldsymbol{z_c} = \boldsymbol{z} - \bar{\boldsymbol{x}}$.

7. Проецирование центрированного вектора признаков $\boldsymbol{z_c}$ в новую систему координат, задаваемую ортонормированными векторами $\{\boldsymbol{\nu_i}\}_{i=1}^{\widehat{n}}$:

$$\boldsymbol{y} = (y_1, \ldots, y_{\widehat{n}})^T = (\boldsymbol{\nu_1}, \ldots, \boldsymbol{\nu_{\widehat{n}}})^T \cdot \boldsymbol{z_c},$$

здесь $y_i = \boldsymbol{\nu_i}^T \cdot \boldsymbol{z_c}$ называется i-ой главной компонентой вектора $\boldsymbol{z}$.

Результаты экспериментов показали, что повторная нормализация после сжатия при помощи метода главных компонент необязательна.

Четвертый этап методики с точки зрения вычислительных ресурсов является наиболее трудоемким и состоит из следующих рекурсивно повторяющихся последовательностей действий: вычисление зависимостей текущего классификатора, формирование входных сигналов для текущего классификатора, обучение текущего классификатора. Была разработана специальная древовидная структура для хранения классификаторов, которая позволяет осуществлять эффективный нисходящий спуск по всем цепочкам зависимостей, начиная с верхнеуровневого классификатора до терминальных узлов, представленных детекторами. Обучение каждого классификатора порождает запрос на обучение нижележащих классификаторов, указанных в списке его зависимостей, и генерацию их выходных данных для формирования входных данных вышележащего классификатора. Следствием используемого таким образом каскадного обучения является возможность «ленивой» загрузки классификаторов: в обучении и распознавании участвуют только те классификаторы, которые напрямую или косвенно встречаются в списке зависимостей классификатора, ответственного за формирование общего решения в коллективе классификационных правил. Это свойство является особенно выгодным при разборе динамических правил обучения классификаторов, т.е. таких правил, от успешного или неуспешного срабатывания которых зависит вызов другого правила. В частности, это характерно для классификационного дерева, когда правила являются вложенными друг в друга. Тем самым за счет применения приема «ленивой» загрузки удается избежать случаев бесполезного вызова того детектора, чье выходное значение, как уже известно, не повлияет на результат общего коллектива классификационных правил.

Пятый этап методики включает в себя два режима: режим оценки эффективности и режим функционирования. В первом режиме осуществляется вычисление показателей оценки качества классификационных моделей, во втором режиме выполняется диагностика системы без априорного знания о фактическом классе идентифицируемого сетевого соединения.

Были выделены следующие показатели оценки качества классификационной модели G:

1. TPR — показатель корректности обнаружения сетевых атак:

$$TPR = \frac{TP}{TP + FN} = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \neq 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \neq 0\right\}_{i=1}^{M^*}},$$

где $TP = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \neq 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}$ — число верно распознанных аномальных соединений, $FN = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \neq 0\right\}_{i=1}^{M^*} - TP$ — число ошибок II рода (число пропусков атаки).

2. FPR — показатель ложных срабатываний:

$$FPR = \frac{FP}{FP + TN} = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i = 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i = 0\right\}_{i=1}^{M^*}},$$

где $FP = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i = 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}$ — число ошибок I рода (число ложных срабатываний), $TN = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i = 0\right\}_{i=1}^{M^*} - FP$ — число верно распознанных нормальных соединений.

3. CCR — показатель корректности классификации соединений:

$$CCR = \frac{CC}{TP + FN + FP + TN} = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \{\bar{c}_i\} = G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)}\right\}_{i=1}^{M^*}},$$

где $CC = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \{\bar{c}_i\} = G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}$ — общее число элементов, класс которых был верно определен, на объединенном наборе данных, состоящем из нормальных и аномальных соединений.

4. ICR — показатель некорректной классификации:

$$ICR = \frac{IC}{TP + FN + FP + TN} = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)}\right\}_{i=1}^{M^*}},$$

где $IC = \#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \mathcal{X}_{\mathcal{C}}^{(TS)} \wedge \bar{c}_i \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{M^*}$ — число случаев некорректной классификации. Отметим, что величина $1 - CCR - ICR$ не обязана быть равной 0 и может быть интерпретирована как показатель конфликтной корректной классификации.

5. GPR — показатель обобщающей способности при обнаружении:

$$GPR = \frac{\overline{TP^*}}{\overline{TP^*} + \overline{FN^*}} = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(TS)} \setminus \mathcal{X}_{\mathcal{C}}^{(LS)} \wedge \bar{c}_i \neq 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{\bar{M}^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(TS)} \setminus \mathcal{X}_{\mathcal{C}}^{(LS)} \wedge \bar{c}_i \neq 0\right\}_{i=1}^{\bar{M}^*}},$$

где показатели $\overline{TP^*}$, $\overline{FN^*}$ представляют собой соответственно число верно распознанных аномальных соединений и число ошибок II рода, которые вычислены на уникальных данных контрольного множества $\bar{\mathcal{X}}_{\mathcal{C}}^{(TS)}$ мощности $\bar{M}^*$, строго исключая любые данные обучающего множества $\mathcal{X}_{\mathcal{C}}^{(LS)}$.

6. OPR — показатель переобученности при обнаружении:

$$OPR = \overline{TPR} - GPR = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(LS)} \wedge \bar{c}_i \neq 0 \wedge 0 \notin G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{\bar{M}}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(LS)} \wedge \bar{c}_i \neq 0\right\}_{i=1}^{\bar{M}}} - GPR,$$

где $\overline{TPR}$ — показатель корректности обнаружения на уникальных данных обучающего множества $\bar{\mathcal{X}}_{\mathcal{C}}^{(LS)}$ мощности $\bar{M}$.

7. GCR — показатель обобщающей способности при классификации:

$$\begin{aligned} GCR &= \frac{\overline{CC^*}}{\overline{TP^*} + \overline{FN^*} + \overline{FP^*} + \overline{TN^*}} = \\ &= \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(TS)} \setminus \mathcal{X}_{\mathcal{C}}^{(LS)} \wedge \{\bar{c}_i\} = G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{\bar{M}^*}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(TS)} \setminus \mathcal{X}_{\mathcal{C}}^{(LS)}\right\}_{i=1}^{\bar{M}^*}}, \end{aligned}$$

где показатели $\overline{CC^*}$, $\overline{FP^*}$, $\overline{TN^*}$ представляют собой соответственно число верно классифицированных соединений, число ошибок I рода и число верно распознанных нормальных соединений, которые вычислены на уникальных данных контрольного множества $\bar{\mathcal{X}}_{\mathcal{C}}^{(TS)}$ мощности $\bar{M}^*$, строго исключая любые данные обучающего множества $\mathcal{X}_{\mathcal{C}}^{(LS)}$.

8. OCR — показатель переобученности при классификации:

$$OCR = \overline{CCR} - GCR = \frac{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(LS)} \wedge \{\bar{c}_i\} = G\left(\boldsymbol{z_i}\right)\right\}_{i=1}^{\bar{M}}}{\#\left\{\boldsymbol{z_i} \,\middle|\, \boldsymbol{z_i} \in \bar{\mathcal{X}}_{\mathcal{C}}^{(LS)}\right\}_{i=1}^{\bar{M}}} - GCR,$$

где $\overline{CCR}$ — показатель корректности классификации на уникальных данных обучающего множества $\bar{\mathcal{X}}_{\mathcal{C}}^{(LS)}$ мощности $\bar{M}$.

Выводы по главе 2

В главе 2 решены следующие задачи:

1. Выполнен анализ моделей искусственных иммунных систем, приведено описание прототипов СОА, построенных на основе искусственных иммунных систем.
2. Представлены модели бинарных классификаторов, описан алгоритм классификации сетевых соединений при помощи них.
3. Разработаны модель искусственной иммунной системы на базе эволюционного подхода, алгоритм генетико-конкурентного обучения сети Кохонена и методика иерархической гибридизации бинарных классификаторов с приложением к обнаружению аномальных сетевых соединений.

Глава 3 Программная реализация СОА и экспериментальная оценка ее эффективности

3.1 Компоненты обнаружения сетевых атак на основе сигнатурного анализа

Сигнатурный анализ как процесс обнаружения сетевых атак включает в себя широкий класс правил, которые направлены на сравнение значений определенных полей (или вычисленных мета-атрибутов) пакетов с рядом сигнатур, заданных пользователем или заложенных статически в систему. Сами сигнатуры могут представляться как (I) ключевые слова (шаблонные подстроки) или (II) команды (функции с аргументами), которые указывают на аномальное действие в сети. В случае I выполняется последовательный просмотр ключевых слов, указанных в сигнатурном правиле, и для каждого из ключевых слов проверяется его посимвольное или побайтовое вхождение в анализируемую строку, содержащуюся в захваченном сетевом пакете. В англоязычной литературе для обозначения процедур поиска такого типа используется термин „сопоставление с образцом“ (pattern matching), частными аналогами которых являются алгоритмы поиска подстроки или регулярные выражения. В случае II сигнатуры обозначаются неявно через результат вызова некоторой системной или пользовательской функции, которая возвращает значение мета-атрибута, подлежащего дальнейшему сравнению. В то время как сигнатуры типа I основаны на простой идее поиска шаблонной подстроки в каждом пакете и требуют от разработчика выбора алгоритма, удовлетворяющего наименьшим временным затратам, сигнатуры типа II, как правило, используются с целью аккумулирования статистических показателей, вычисленных для нескольких определенных пакетов или соединений. В данном разделе проведено исследование сигнатур типа I, а именно нескольких алгоритмов поиска шаблонной подстроки, которые являются наиболее распространенными при реализации сигнатурных СОА. Примеры сигнатур типа II для обнаружения атак, направленных на подбор пароля, приведены в приложении Б.

Для проведения экспериментов, связанных с анализом сигнатурного механизма в СОА, было выбрано три различных алгоритма поиска шаблонной подстроки: алгоритм Ахо—Корасик [45, 46, 194], алгоритм Бойера—Мура [45, 63, 194] и классический инкрементальный алгоритм (Linux man strstr); и реализованы их различные параллельные модификации.

Первый алгоритм заключается в построении детерминированного конечного автомата, который рассматривается как некоторое компактное представление множества искомых шаблонных подстрок $keywords$. Алгоритм инициализации автомата $acfsm$ представлен в алгоритме 13.

Алгоритм 13: Построение конечного автомата в алгоритме Ахо—Корасик

```
   Входные данные: Шаблонные подстроки keywords
   Выходные данные: Автомат acfsm
 1 memset(acfsm.out, 0, sizeof(acfsm.out))          // бинарная таблица совпадения подстрок
 2 memset(acfsm.fail, −1, sizeof(acfsm.fail))  // массив переходов по наибольшему суффиксу
 3 memset(acfsm.goto, −1, sizeof(acfsm.goto))     // таблица переходов по текущему символу
 4 memset(acfsm.depth, 0, sizeof(acfsm.depth))       // массив глубин для каждого состояния
 5 states := 1
 6 для каждого keyword ∈ keywords выполнять
 7     cur_state := 0
 8     cur_depth := 0
 9     цикл i от 0 до strlen(keyword) − 1 выполнять
10         c := ascii_num(keyword[i])                                          // c ∈ {0,...,N − 1}
11         если acfsm.goto[cur_state][c] = −1 тогда
12             acfsm.goto[cur_state][c] := states + +              // добавить ребро в автомат
13         cur_state := acfsm.goto[cur_state][c]
14         acfsm.depth[cur_state] := + + cur_depth
15 цикл i от 0 до sizeof(acfsm.goto[0]) − 1 выполнять
16     если acfsm.goto[0][i] = −1 тогда
17         acfsm.goto[0][i] := 0                          // создать петлю в корне автомата
18 queue := create_fifo()                                          // создать очередь FIFO
   /* пройти все символы (под-)алфавита размера N                                    */
19 цикл c от 0 до N − 1 выполнять
20     если acfsm.goto[0][c]! = 0 тогда
21         acfsm.fail[acfsm.goto[0][c]] := 0
22         push_back(queue,acfsm.goto[0][c])            // добавить элемент в хвост очереди
23 пока !empty(queue) выполнять
24     state := pop_front(queue)                        // извлечь элемент из головы очереди
25     цикл c от 0 до N − 1 выполнять
26         если acfsm.goto[state][c]! = −1 тогда
27             failure := acfsm.fail[state]
               /* инициализировать массив переходов по наибольшему суффиксу     */
28             пока acfsm.goto[failure][c] = −1 выполнять
29                 failure := acfsm.fail[failure]
30             failure := acfsm.goto[failure][c]
31             acfsm.fail[acfsm.goto[state][c]] := failure
32             цикл i от 0 до sizeof(acfsm.out[failure]) выполнять
33                 если acfsm.out[failure][i] = 1 тогда
34                     acfsm.out[acfsm.goto[state][c]][i] := acfsm.out[failure][i]
35         push_back(queue,acfsm.goto[state][c])
```

Алгоритм поиска подстрок $keywords$ в анализируемой строке $string$ представлен в алгоритме 14.

Алгоритм 14: Поиск подстрок в алгоритме Ахо—Корасик

Входные данные: Анализируемая строка $string$, автомат $acfsm$, шаблонные подстроки $keywords$
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку $occurrences$

```
1  memset(zero_block,0,sizeof(zero_block))  // массив, sizeof(zero_block)=sizeof(keywords)
2  memset(occurrences,0,sizeof(occurrences))
3  memset(key_lens,0,sizeof(key_lens))      // массив, sizeof(key_lens)=sizeof(keywords)
4  цикл i от 0 до sizeof(key_lens) − 1 выполнять
5      key_lens[i] := strlen(keywords[i])
6  str_len := strlen(string)
7  cur_state := 0
8  цикл i от 0 до str_len − 1 выполнять
9      c := ascii_num(string[i])
10     пока acfsm.goto[cur_state][c] = −1 выполнять
11         cur_state := acfsm.fail[cur_state]
12     cur_state := acfsm.goto[cur_state][c]
13     если memcmp(acfsm.out[cur_state],zero_block,sizeof(zero_block)) = 0 тогда
14         continue
15     цикл j от 0 до sizeof(keywords) − 1 выполнять
16         если acfsm.out[cur_state][j] = 1 тогда
               /* подстрока keywords[j] найдена в строке string                  */
               /* с позиции i - key_lens[j] + 1 по позицию i                     */
17             occurrences[j] := 1                              // признак вхождения
```

Для ускорения работы алгоритма Ахо—Корасик был реализован алгоритм параллельной обработки анализируемой строки $string$. Данная модификация заключается в разбиении такой строки на несколько кусков, каждый из которых обрабатывается независимо несколькими потоками. В данном подходе конечный автомат запускается в нескольких копиях, каждая их которых сканирует входную строку, начиная с жестко определенной для него позиции (левой границы). Тонкой особенностью в этой параллельной модификации является то, что, в отличие от левой границы, правая граница является „плавающей“. Здесь возникает необходимость обработки перекрывающихся частей отдельных кусков входной строки в случае, когда длина шаблонной подстроки превосходит длину оставшейся для сканирования части обрабатываемого куска входной строки. Поэтому для этого алгоритма следует отслеживать изменение значений глубин в дереве (конечном автомате) при неуспешном переходе в следующее состояние по текущему наблюдаемому символу. В противном случае возможны случаи пропусков.

Этот подход был позаимствован из работы [147], авторы которой получили ускорение на GPU, почти в 4000 раз превосходящее однопоточную реализацию этого же алгоритма. В алгоритмах 15 и 16 приведены псевдокоды такой параллельной модификации для алгоритма Ахо–Корасик с использованием OpenMP и CUDA.

Алгоритм 15: Поиск подстрок в алгоритме Ахо–Корасик (CPU OpenMP)

```
   Входные данные: Анализируемая строка string, автомат acfsm, шаблонные подстроки keywords
   Выходные данные: Массив признаков вхождения подстрок в анализируемую строку occurrences
1  memset(zero_block,0,sizeof(zero_block)) // массив, sizeof(zero_block)=sizeof(keywords)
2  memset(occurrences,0,sizeof(occurrences))
3  memset(key_lens,0,sizeof(key_lens))       // массив, sizeof(key_lens)=sizeof(keywords)
4  цикл i от 0 до sizeof(key_lens) − 1 выполнять
5      key_lens[i] := strlen(keywords[i])
6  str_len := strlen(string)
7  n_threads := 8                                                    // число потоков
8  memset(chunks,0,sizeof(chunks))                  // массив, sizeof(chunks)=n_threads+1
9  цикл p от 1 до sizeof(chunks) − 1 выполнять
10     chunks[p] := chunks[p − 1] + str_len/n_threads
11     если p <= str_len%n_threads тогда
12         + + chunks[p]
13 #pragma omp parallel num_threads(n_threads)
14 начало блока
15     #pragma omp for
16     цикл p от 0 до n_threads − 1 выполнять
17         cur_state := 0
18         prev_depth := 0                                  // предыдущее значение глубины
19         k := 0
20         цикл i от chunks[p] до str_len − 1 выполнять
21             c := ascii_num(string[i])
22             пока acfsm.goto[cur_state][c] = −1 выполнять
23                 cur_state := acfsm.fail[cur_state]
24             cur_state := acfsm.goto[cur_state][c]
25             cur_depth := acfsm.depth[cur_state]
26             если cur_depth <= prev_depth тогда
27                 k+ = prev_depth − cur_depth + 1
28             если k >= chunks[p + 1] − chunks[p] тогда
29                 break
30             prev_depth := cur_depth
31             если memcmp(acfsm.out[cur_state],zero_block,sizeof(zero_block)) = 0 тогда
32                 continue
33             цикл j от 0 до sizeof(keywords) − 1 выполнять
34                 если acfsm.out[cur_state][j] = 1&&i − key_lens[j] + 1 >= chunks[p]&&
                    i − key_lens[j] + 1 < chunks[p + 1] тогда
                       /* подстрока keywords[j] найдена в строке string          */
                       /* с позиции i - key_lens[j] + 1 по позицию i             */
35                     occurrences[j] := 1                          // признак вхождения
```

Алгоритм 16: Поиск подстрок в алгоритме Ахо–Корасик (GPU CUDA)

Входные данные: Анализируемая строка $string$, автомат $acfsm$, шаблонные подстроки $keywords$
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку $occurrences$

```
memset(occurrences,0,sizeof(occurrences))
memset(key_lens,0,sizeof(key_lens))        // массив, sizeof(key_lens)=sizeof(keywords)
цикл i от 0 до sizeof(key_lens) − 1 выполнять
    key_lens[i] := strlen(keywords[i])
str_len := strlen(string)
cudaMalloc(&dev_acfsm, sizeof(acfsm))
cudaMemcpy(dev_acfsm, acfsm, sizeof(acfsm), cudaMemcpyHostToDevice)
cudaMalloc(&dev_string, str_len)                  // строка без терминального символа '\0'
cudaMemcpy(dev_string, string, str_len, cudaMemcpyHostToDevice)
cudaMalloc(&dev_keywords, sizeof(keywords[0]) * sizeof(keywords))
memset(aux, 0, sizeof(aux))                                  // вспомогательный массив
цикл i от 0 до sizeof(keywords) − 1 выполнять
    cudaMalloc(&aux[i], key_lens[i])
    cudaMemcpy(aux[i], keywords[i], key_lens[i], cudaMemcpyHostToDevice)
cudaMemcpy(dev_keywords,aux,sizeof(keywords[0]) * sizeof(keywords),cudaMemcpyHostToDevice)
cudaMalloc(&dev_key_lens, sizeof(key_lens))
cudaMemcpy(dev_key_lens,key_lens,sizeof(key_lens),cudaMemcpyHostToDevice)
cudaMalloc(&dev_occurrences,sizeof(occurrences))
cudaMemset(dev_occurrences,0,sizeof(occurrences))
block_size := 1024
n_blocks := 4
n_gpu_tasks := n_blocks * block_size                                  // 4096 потоков
find_substrs <<< n_blocks, block_size >>> (dev_acfsm, dev_string, str_len,
  dev_keywords, sizeof(keywords),dev_key_lens, n_gpu_tasks,dev_occurrences)  // запустить ядро
cudaMemcpy(occurrences,dev_occurrences,sizeof(occurrences),cudaMemcpyDeviceToHost)
cudaFree(dev_acfsm)
цикл i от 0 до sizeof(keywords) − 1 выполнять
    cudaFree(aux[i])
cudaFree(dev_keywords)
cudaFree(dev_key_lens)
cudaFree(dev_string)
cudaFree(dev_occurrences)
__global__ void find_substrs(dev_acfsm,dev_string,dev_str_len,dev_keywords,dev_keys_size,
  dev_key_lens,n_gpu_tasks,dev_occurrences)                                  // kernel
начало блока
    idx := blockIdx.x * blockDim.x + threadIdx.x
    from_pos := (dev_str_len/n_gpu_tasks) * idx
    если idx < dev_str_len%n_gpu_tasks тогда
        from_pos+ = idx
    иначе
        from_pos+ = dev_str_len%n_gpu_tasks
    to_pos = from_pos + dev_str_len/n_gpu_tasks + (idx < dev_str_len%n_gpu_tasks?1 : 0)
    cur_state := prev_depth := k := 0
    цикл i от from_pos до dev_str_len − 1 выполнять
        c := ascii_num(string[i])
        пока acfsm.goto[cur_state][c] = −1 выполнять
            cur_state := acfsm.fail[cur_state]
        cur_state := acfsm.goto[cur_state][c]
        cur_depth := acfsm.depth[cur_state]
        если cur_depth <= prev_depth тогда
            k+ = prev_depth − cur_depth + 1
        если k >= to_pos − from_pos тогда
            break
        prev_depth := cur_depth
        цикл j от 0 до dev_keys_size − 1 выполнять
            если acfsm.out[cur_state][j] = 1&&i − dev_key_lens[j] + 1 >= from_pos&&
              i − dev_key_lens[j] + 1 < to_pos тогда
                /* подстрока keywords[j] найдена в строке string                */
                /* с позиции i - key_lens[j] + 1 по позицию i                   */
                dev_occurrences[j] := 1                              // признак вхождения
```

Алгоритм Бойера–Мура основан на построении двух таблиц $good\_suffs$ и bad_chars, которые используются для сдвига входной строки $string$ на одну или несколько позиций вправо. В алгоритме 17 приведен псевдокод предварительной настройки такого поиска, включающей инициализацию этих таблиц.

Алгоритм 17: Предварительная обработка шаблонных подстрок в алгоритме Бойера–Мура

```
   Входные данные: Шаблонные подстроки keywords
   Выходные данные: Таблицы good_suffs и bad_chars
1  цикл i от 0 до sizeof(keywords) − 1 выполнять
2      key_len ≔ strlen(keywords[i])
3      memset(suffixes,0,sizeof(suffixes))                    // sizeof(suffixes)>=key_len
4      l ≔ 0
5      r ≔ key_len − 1
6      suffixes[r] ≔ key_len
7      цикл j от key_len − 2 до 0 с шагом −1 выполнять
8          если j > r&&suffixes[j + key_len − 1 − l] < j − r тогда
9              suffixes[j] ≔ suffixes[j + key_len − 1 − l]
10         иначе
11             если j < r тогда
12                 r ≔ j
13             l ≔ j
14             пока r >= 0&&keywords[i][r] = keywords[i][r − key_len − 1 − l] выполнять
15                 − − r
16             suffixes[j] ≔ l − r
17     цикл j от 0 до key_len − 1 выполнять
18         good_suffs[i][j] ≔ key_len
19     k ≔ 0
20     цикл j от key_len − 1 до 0 с шагом −1 выполнять
21         если suffixes[j] = j + 1 тогда
22             пока k < key_len − 1 − j выполнять
23                 если good_suffs[i][k] ≔ key_len тогда
24                     good_suffs[i][k] ≔ key_len − 1 − j
25                 + + k
26     цикл j от 0 до key_len − 2 выполнять
27         good_suffs[i][key_len − 1 − suffixes[j]] ≔ key_len − 1 − j
28     цикл j от 0 до sizeof(bad_chars) − 1 выполнять
29         bad_chars[i][j] ≔ key_len
30     цикл j от 0 до key_len − 2 выполнять
31         bad_chars[i][ascii_num(keywords[i][j])] ≔ key_len − j − 1
```

Сравнение входной строки $string$ с одной из шаблонных подстрок $keywords$ в данном алгоритме осуществляется в направлении справа налево. Величина сдвига задается при помощи таблиц $good\_suffs$ и bad_chars. В

алгоритме 18 приведен псевдокод поиска шаблонных подстрок $keywords$ внутри анализируемой строки $string$ при помощи алгоритма Бойера–Мура.

Алгоритм 18: Поиск подстрок в алгоритме Бойера–Мура

```
Входные данные: Анализируемая строка string, таблицы good_suffs и bad_chars, подстроки keywords
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку occurrences
memset(occurrences,0,sizeof(occurrences))
memset(key_lens,0,sizeof(key_lens))        // массив, sizeof(key_lens)=sizeof(keywords)
цикл i от 0 до sizeof(key_lens) − 1 выполнять
    key_lens[i] := strlen(keywords[i])
str_len := strlen(string)
цикл i от 0 до sizeof(keywords) − 1 выполнять
    j := 0
    пока j <= str_len − key_lens[i] выполнять
        k := key_lens[i] − 1
        пока k >= 0&&keywords[i][k] = string[j + k] выполнять
            − − k
        если k < 0 тогда
            /* подстрока keywords[i] найдена в строке string                    */
            /* с позиции j по позицию j+key_lens[i]-1                           */
            occurrences[i] := 1                                  // признак вхождения
        иначе
            j+ = max(good_suffs[i][k],bad_chars[i][ascii_num(string[j + k])] − key_lens[i] + 1 + k)
```

Для ускорения работы данного алгоритма была разработана его модифицированная версия, которая использует многопоточное распараллеливание операций поиска на уровне множества искомых шаблонных подстрок. С этой целью это множество разбивается в примерно равных пропорциях, и каждое из образовавшихся подмножеств множества $keywords$ назначается отдельному потоку с целью поиска его элементов в сканируемой строке $string$. По сравнению с предыдущей версией параллельной модификации, использовавшейся для оптимизации алгоритма Ахо–Корасик, эту версию отличает простота в реализации. Кроме того, при наличии слишком большого множества $keywords$ выигрыш от использования такой модификации становится существенным.

В алгоритме 19 приведен псевдокод поиска подстрок в алгоритме Бойера–Мура при помощи вышепредставленной оптимизации и технологии OpenMP. Изменения данного алгоритма по сравнению с 18 минимальны: добавлена пара директив препроцессора omp. В остальном OpenMP берет на себя задачу „раскидывания“ кусков массива $keywords$ между потоками, равномерно их загружая,

что освобождает программиста от ручного кодирования для привязки определенных участков массива к каждому потоку.

Алгоритм 19: Поиск подстрок в алгоритме Бойера–Мура (CPU OpenMP)

Входные данные: Анализируемая строка $string$, таблицы $good\_suffs$ и bad_chars, подстроки $keywords$
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку $occurrences$

```
1  memset(occurrences,0,sizeof(occurrences))
2  memset(key_lens,0,sizeof(key_lens))        // массив, sizeof(key_lens)=sizeof(keywords)
3  цикл i от 0 до sizeof(key_lens) − 1 выполнять
4      key_lens[i] := strlen(keywords[i])
5  str_len := strlen(string)
6  n_threads := 8                                                    // число потоков
7  #pragma omp parallel num_threads(n_threads)
8  начало блока
9      #pragma omp for
10     цикл i от 0 до sizeof(keywords) − 1 выполнять
11         j := 0
12         пока j <= str_len − key_lens[i] выполнять
13             k := key_lens[i] − 1
14             пока k >= 0&&keywords[i][k] = string[j + k] выполнять
15                 − − k
16             если k < 0 тогда
                   /* подстрока keywords[i] найдена в строке string              */
                   /* с позиции j по позицию j+key_lens[i]-1                     */
17                 occurrences[i] := 1                                // признак вхождения
18             иначе
19                 j+ = max(good_suffs[i][k],bad_chars[i][ascii_num(string[j+k])]−key_lens[i]+1+k)
```

В алгоритме 20 приведен псевдокод поиска подстрок в алгоритме Бойера–Мура при помощи вышепредставленной оптимизации и технологии CUDA. Изменения по сравнению с алгоритмом 18 коснулись добавления ядра (функции, выполняющейся на видеокарте) и закрепления за каждой нитью с идентификатором idx среза массива $keywords[from\_pos, \ldots, to\_pos - 1]$. Здесь, как и в алгоритме 16, используются 4096 потоков, на вход каждого из которых поступают идентичные данные согласно принципу вычислений SIMD (Single Instruction Multiple Data). Управление потоками и разделение между ними непересекающихся подзадач осуществляется через индексы потоков $threadIdx.x$ и индексы объемлющих их блоков $blockIdx.x$. После вызова ядра выполняется копирование бинарного массива $dev\_occurrences$ на хостовое устройство (оперативную память). Единичные элементы этого массива с индексами i указывают на признак вхождения подстроки $keywords[i]$ в анализируемую строку $string$.

Алгоритм 20: Поиск подстрок в алгоритме Бойера–Мура (GPU CUDA)

Входные данные: Анализируемая строка *string*, таблицы *good_suffs* и *bad_chars*, подстроки *keywords*
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку *occurrences*

```
memset(occurrences,0,sizeof(occurrences))
memset(key_lens,0,sizeof(key_lens))        // массив, sizeof(key_lens)=sizeof(keywords)
цикл i от 0 до sizeof(key_lens) − 1 выполнять
    key_lens[i] := strlen(keywords[i])
str_len := strlen(string)
cudaMalloc(&dev_good_suffs,sizeof(good_suffs))
cudaMemcpy(dev_good_suffs, good_suffs, sizeof(good_suffs), cudaMemcpyHostToDevice)
cudaMalloc(&dev_bad_chars,sizeof(bad_chars))
cudaMemcpy(dev_bad_chars, bad_chars, sizeof(bad_chars), cudaMemcpyHostToDevice)
cudaMalloc(&dev_string, str_len)              // строка без терминального символа '\0'
cudaMemcpy(dev_string, string, str_len, cudaMemcpyHostToDevice)
cudaMalloc(&dev_keywords, sizeof(keywords[0]) * sizeof(keywords))
memset(aux, 0, sizeof(aux))                                  // вспомогательный массив
цикл i от 0 до sizeof(keywords) − 1 выполнять
    cudaMalloc(&aux[i], key_lens[i])
    cudaMemcpy(aux[i], keywords[i], key_lens[i], cudaMemcpyHostToDevice)
cudaMemcpy(dev_keywords,aux,sizeof(keywords[0]) * sizeof(keywords),cudaMemcpyHostToDevice)
cudaMalloc(&dev_key_lens, sizeof(key_lens))
cudaMemcpy(dev_key_lens,key_lens,sizeof(key_lens),cudaMemcpyHostToDevice)
cudaMalloc(&dev_occurrences,sizeof(occurrences))
cudaMemset(dev_occurrences,0,sizeof(occurrences))
block_size := 1024
n_blocks := 4
n_gpu_tasks := n_blocks * block_size                                  // 4096 потоков
search <<< n_blocks, block_size >>> (dev_good_suffs,dev_bad_suffs,dev_string,str_len,
  dev_keywords,sizeof(keywords),dev_key_lens,n_gpu_tasks,dev_occurrences)   // запустить ядро
cudaMemcpy(occurrences,dev_occurrences,sizeof(occurrences),cudaMemcpyDeviceToHost)
cudaFree(dev_acfsm)
цикл i от 0 до sizeof(keywords) − 1 выполнять
    cudaFree(aux[i])
cudaFree(dev_keywords)
cudaFree(dev_key_lens)
cudaFree(dev_string)
cudaFree(dev_occurrences)
__global__ void search(dev_good_suffs,dev_bad_suffs,dev_string,dev_str_len,
  dev_keywords,dev_keys_size,dev_key_lens,n_gpu_tasks,dev_occurrences)        // kernel
начало блока
    idx := blockIdx.x * blockDim.x + threadIdx.x
    from_pos := (dev_keys_size/n_gpu_tasks) * idx
    если idx < dev_keys_size%n_gpu_tasks тогда
        from_pos+ = idx
    иначе
        from_pos+ = dev_keys_size%n_gpu_tasks
    to_pos = from_pos + dev_keys_size/n_gpu_tasks + (idx < dev_keys_size%n_gpu_tasks?1 : 0)
    цикл i от from_pos до to_pos − 1 выполнять
        j := 0
        пока j <= dev_str_len − dev_key_lens[i] выполнять
            k := dev_key_lens[i] − 1
            пока k >= 0&&dev_keywords[i][k] = dev_string[j + k] выполнять
                − − k
            если k < 0 тогда
                /* подстрока keywords[i] найдена в строке string              */
                /* с позиции j по позицию j+key_lens[i]-1                     */
                dev_occurrences[i] := 1                          // признак вхождения
            иначе
                j+ = max(dev_good_suffs[i][k],
                  dev_bad_chars[i][ascii_num(dev_string[j + k])] − dev_key_lens[i] + 1 + k)
```

Последний из рассматривающихся в данном разделе алгоритмов — это классический инкрементальный поиск, являющийся аналогом вызывающейся в цикле функции strstr в языке программирования C. На каждой итерации происходит сдвиг шаблонной подстроки ровно на один символ вправо, и сравнение строк выполняется посимвольно при проходе слева направо. Псевдокод этого поиска приведен в алгоритме 21. В отличие от предыдущих двух алгоритмов, этот алгоритм выделяется наглядной простотой и не использует предварительных сведений о структуре шаблонных подстрок, за счет чего обладает бóльшими временными затратами.

Алгоритм 21: Поиск подстрок в инкрементальном алгоритме

Входные данные: Анализируемая строка $string$, шаблонные подстроки $keywords$
Выходные данные: Массив признаков вхождения подстрок в анализируемую строку $occurrences$

```
1  memset(occurrences,0,sizeof(occurrences))
2  memset(key_lens,0,sizeof(key_lens))        // массив, sizeof(key_lens)=sizeof(keywords)
3  цикл i от 0 до sizeof(key_lens) − 1 выполнять
4      key_lens[i] := strlen(keywords[i])
5  str_len := strlen(string)
6  цикл i от 0 до sizeof(keywords) − 1 выполнять
7      j := 0
8      пока str_len − j >= key_lens[i] выполнять
9          k := 0
10         пока k < key_lens[i]&&string[j + k] = keywords[k] выполнять
11             ++k
12         если k = key_lens[i] тогда
               /* подстрока keywords[i] найдена в строке string                */
               /* с позиции j по позицию j+key_lens[i]-1                       */
13             occurrences[i] := 1                                  // признак вхождения
14         ++j
```

С целью формирования набора тестовых данных, содержащего сканируемые строки ($string$), был реализован генератор символьных последовательностей, суть работы которого заключалась в последовательной конкатенации ряда произвольно сгенерированных символов и произвольно выбранных шаблонных подстрок и добавлении этой записи в набор данных. Основой для наполнения множества шаблонных подстрок ($keywords$) послужили значения атрибута content, извлеченные из сигнатурных правил Snort. На рисунке 3.1 изображены графики, отражающие зависимость времени выполнения каждого из рассмотренных алгоритмов в зависимости от размера множества шаблонных подстрок

при фиксированной длине входной строки в 5000 символов. На рисунке 3.2 показаны кривые, которые соответствуют временным затратам каждого из алгоритмов при изменении длины анализируемой строки для фиксированного размера набора искомых шаблонных подстрок (5000 подстрок).

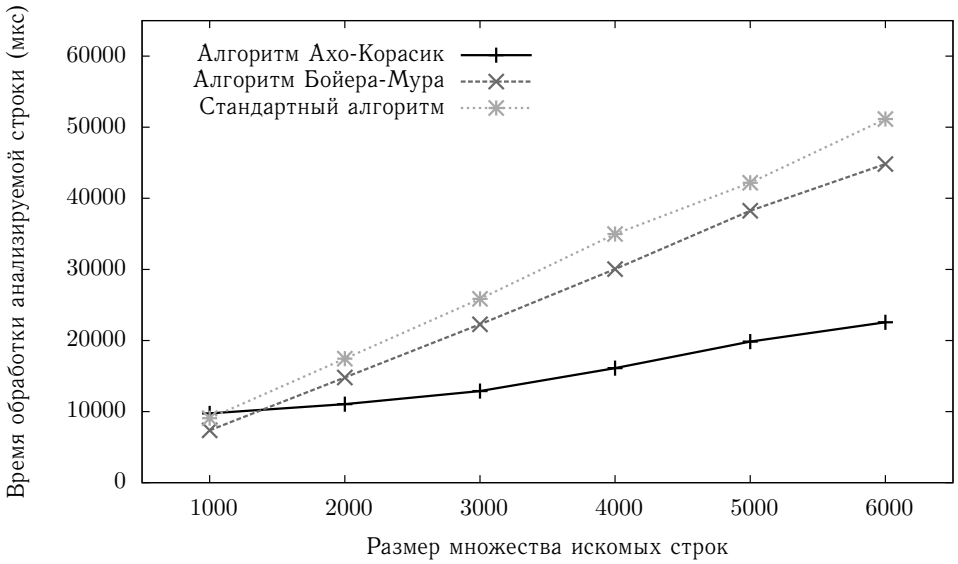


Рисунок 3.1 — Зависимость времени работы алгоритмов поиска шаблонной подстроки от размера множества искомых строк

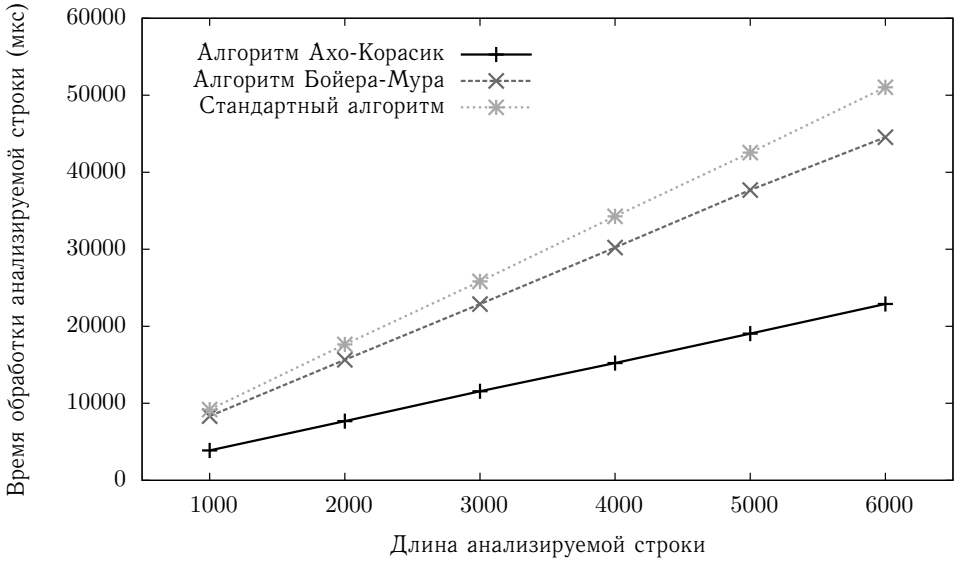


Рисунок 3.2 — Зависимость времени работы алгоритмов поиска шаблонной подстроки от длины анализируемой строки

Эти и последующие данные получены в результате усреднения времени работы алгоритмов при прогоне на ста произвольно сгенерированных строках. Отметим, что в обоих случаях для стандартного алгоритма и алгоритма Бойера—

Мура наблюдается примерно одинаковая асимптотика роста временной сложности. За счет более выгодного хранения шаблонных подстрок в виде конечного автомата алгоритм Ахо—Корасик показывает наилучшие результаты.

В графиках, представленных на рисунках 3.3 и 3.4, показано изменение времени работы модифицированного алгоритма Ахо—Корасик при распараллеливании на уровне CPU и GPU.

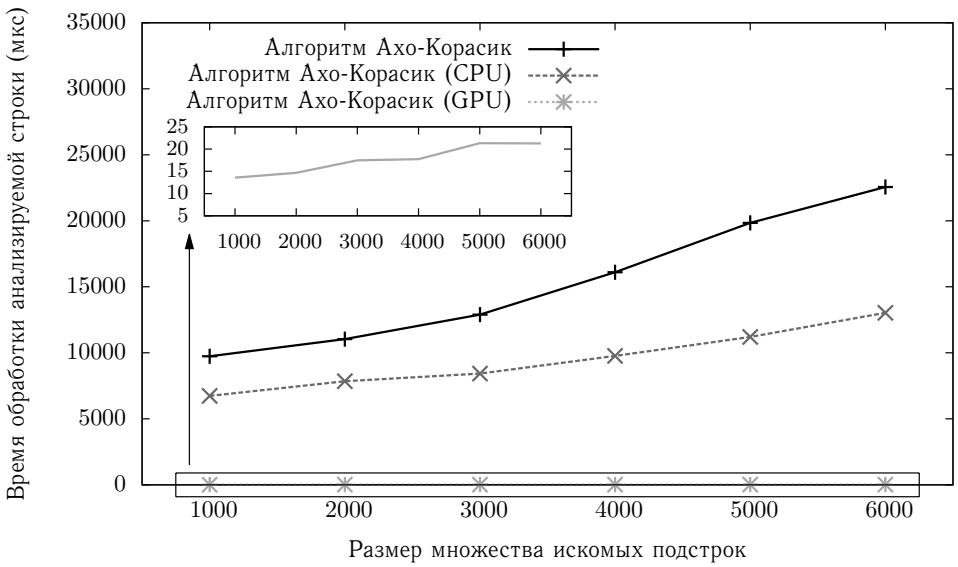


Рисунок 3.3 — Зависимость времени работы алгоритма Ахо—Корасик от размера множества искомых строк

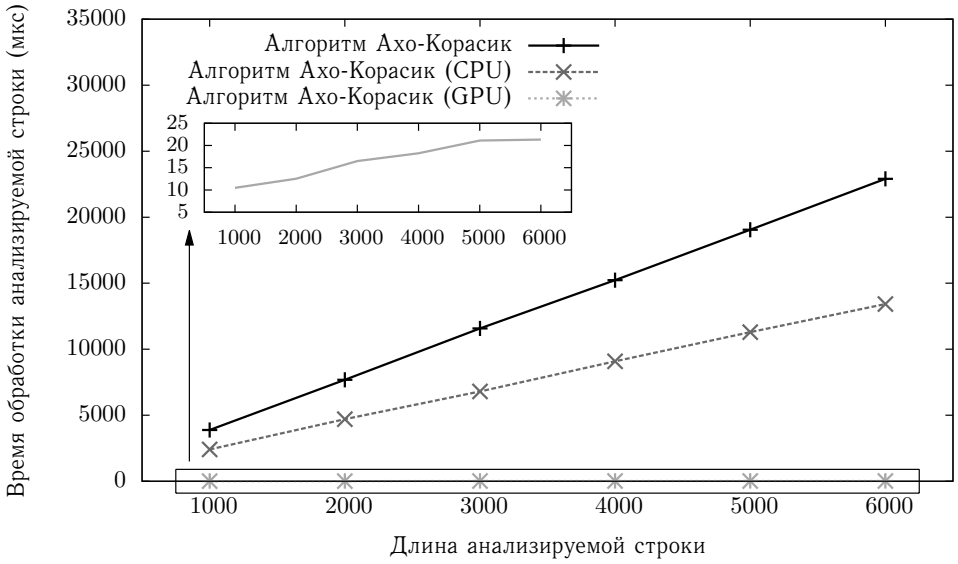


Рисунок 3.4 — Зависимость времени работы алгоритма Ахо—Корасик от длины анализируемой строки

В экспериментах, соответствующих рисункам 3.5 и 3.6, использован подход с применением модификации алгоритма Бойера—Мура.

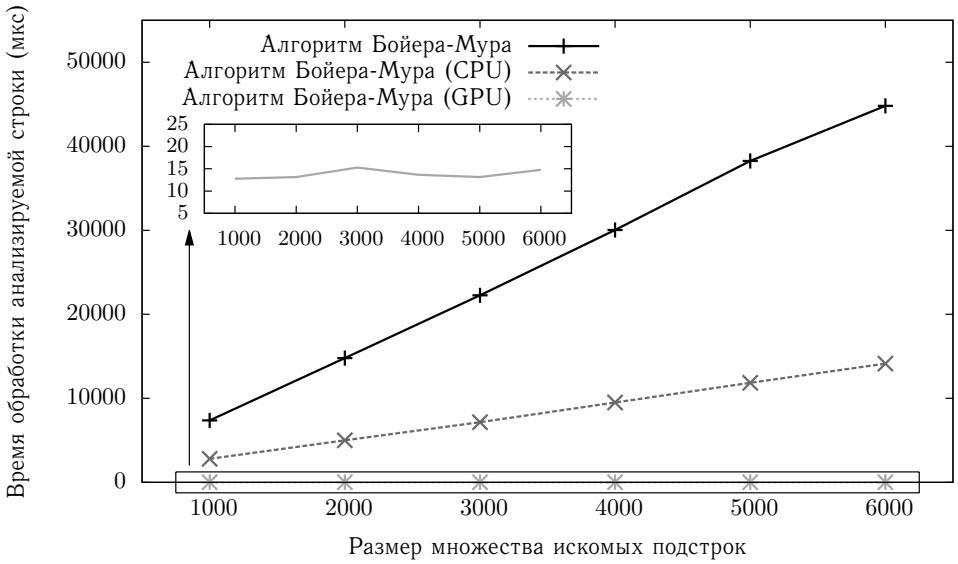


Рисунок 3.5 — Зависимость времени работы алгоритма Бойера—Мура от размера множества искомых строк

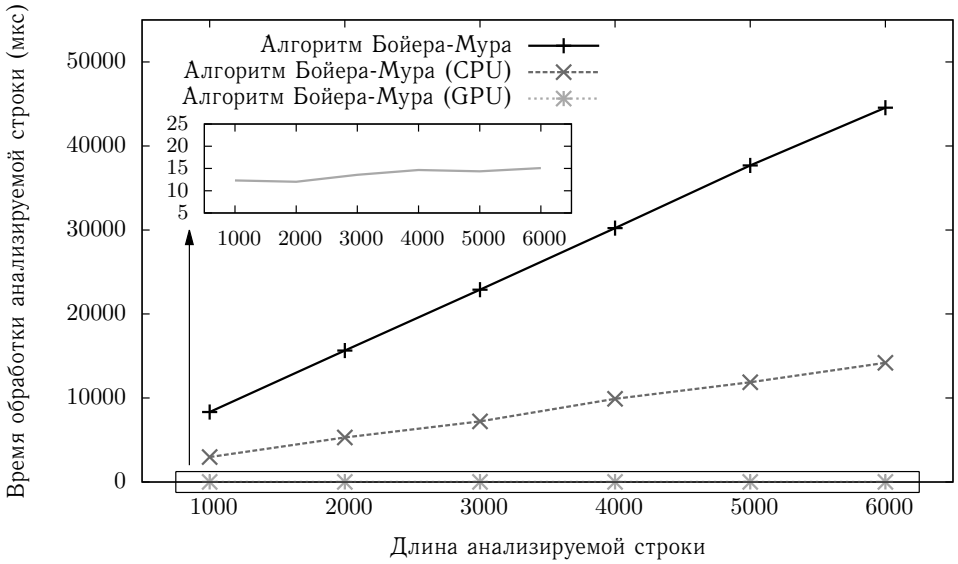


Рисунок 3.6 — Зависимость времени работы алгоритма Бойера—Мура от длины анализируемой строки

В описанных выше подходах для акселерации поиска подстроки использовалось восемь потоков, запущенных на четырех ядрах CPU (2 физических ядра и два виртуальных ядра), и 4096 потоков, запущенных на GPU (четыре блока размером 1024 потока). Характеристики рабочей машины следующие: CPU Intel(R) Core(TM) i5-3210M 2.50Ghz (2 cores), Intel Turbo Boost 2.0 (3.10GHz), Intel

Hyper-Threading, 512KB L2-cache, 3MB L3-cache; RAM 4GB DDR3 1600MHz; HDD 500 GB; GPU NVIDIA GeForce GT 620M 2GB memory. Путем применения этих подходов удалось снизить временные затраты в функционировании алгоритма Бойера—Мура в 2.63–3.23 раза при запуске на CPU и 576.65–3034 раза при запуске на GPU и в работе алгоритме Ахо—Корасик в 1.41–1.77 раза при запуске на CPU и 370.7–1074.38 раза при запуске на GPU. Значения этих величин соответствуют наименьшему и наибольшему приросту производительности каждого из алгоритмов на шести контрольных точках, взятых с шагом измерения 1000. Дальнейшее увеличение размера множества шаблонных подстрок уже приводит к переполнению внутренней памяти видеокарты.

Подчеркнем, что для GPU замеры производились без учета времени, расходуемого на выделение памяти для размещения входной строки (вызов функции cudaMalloc) и ее копирование (вызов функции cudaMemcpy) во внутреннюю память видеокарты. Если вызова первой функции можно в большинстве случаев избежать посредством предварительной аллокации куска памяти достаточно большого размера, то функцию копирования с физической памяти хоста на память устройства приходится вызывать каждый раз для помещения в нее анализируемой строки. С учетом времени, необходимого для вызова этой функции, время работы параллельного алгоритма на GPU становится сравнимым с временем работы ее однопоточной версии. Кроме того, как видно из вышеприведенных алгоритмов, из всех критически важных участков кода исключены лишние строковые операции. В частности, инициализация размеров анализируемой строки и шаблонных подстрок вынесена за пределы основного цикла и тела ядра.

Каждый из рассмотренных алгоритмов реализован автором настоящего диссертационного исследования с использованием языка программирования C без применения сторонних библиотек (за исключением libc.so). Для реализации параллельных модификаций этих алгоритмов были использованы компилятор gcc 4.9.2-10, deb-пакеты libgomp1 4.9.2-10 (OpenMP), nvidia-cuda-dev 6.0.37-5, nvidia-cuda-toolkit 6.0.37-5 (CUDA).

3.2 Архитектура и программная реализация распределенной СОА

В данном разделе представлена архитектура разработанной СОА, в функционирование которой внедрены представленные в главе 2 модели, алгоритмы и методика. Архитектура этой СОА представлена на рисунке 3.7.

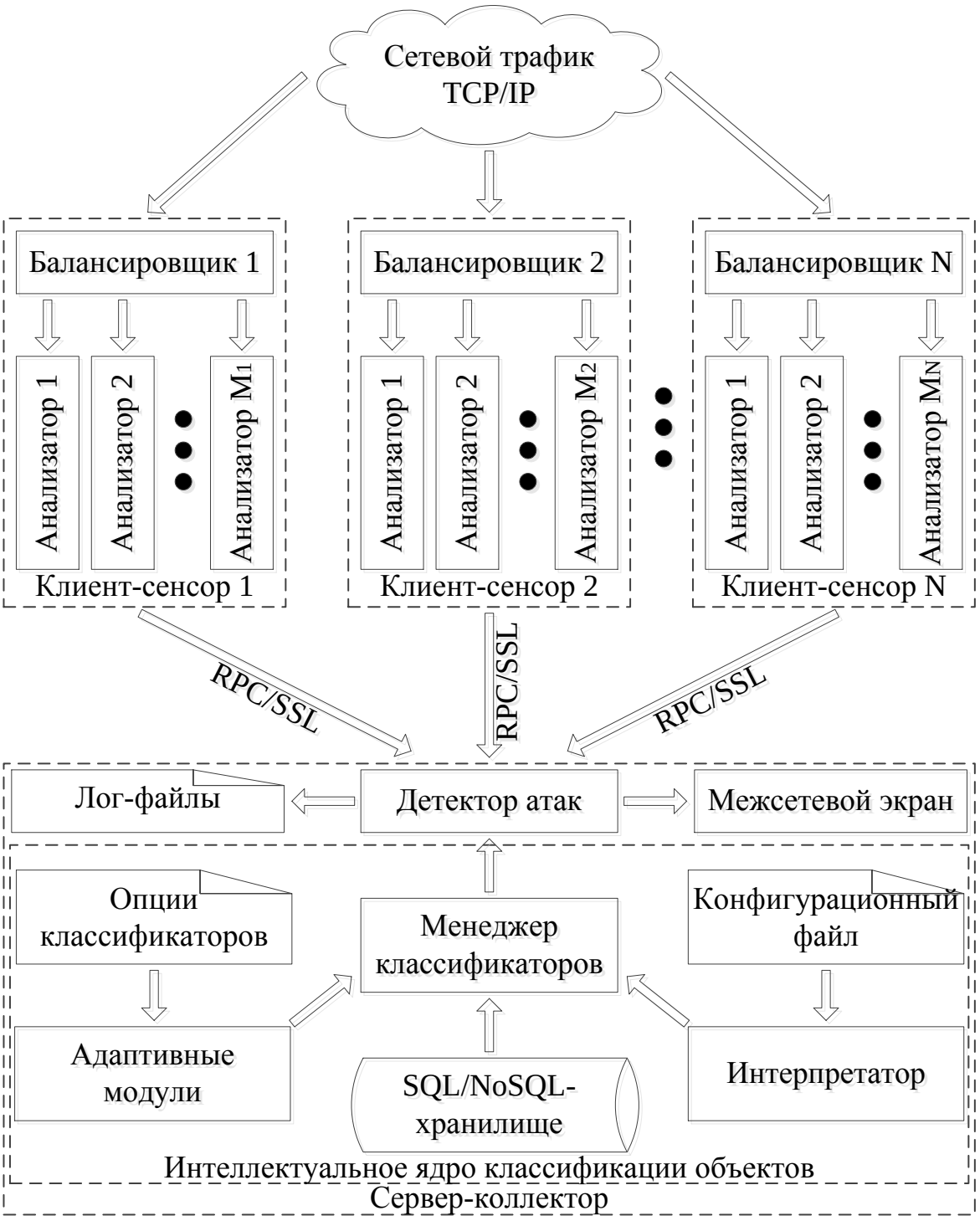


Рисунок 3.7 — Архитектура разработанной СОА

Разработанная СОА является распределенной и состоит из нескольких клиент-сенсоров и одного сервер-коллектора. На каждый контролируемый узел в сети устанавливается два программных компонента: балансировщик трафика и сетевой анализатор. Роль первого заключается в распределении нагрузки между несколькими внутренними фиктивными интерфейсами ОС. Алгоритм функционирования пакетного балансировщика и его экспериментальное исследование приведены в разделе А.10 приложения А. Функции анализатора — дефрагментация IP-пакетов [34, 35], формирование TCP-сессий (TCP-реассемблирование), выделение атрибутов сетевого трафика, включая сбор статистических показателей по входящим IP-адресам, измерение интенсивности приема специальных пакетов. Кроме того, в число задач этого компонента добавлено обнаружение явных нарушений на уровне сети (выявление land-атак, аномальных сетевых пакетов и подозрительных исходящих адресов), проверка целостности пакетов посредством вычисления их контрольных сумм, выполнение сигнатурного анализа содержимого отдельных и дефрагментированных пакетов, а также контента реассемблированных TCP-сегментов с использованием алгоритмов, представленных в разделе 3.1. Разработанные и встроенные в анализатор алгоритмы дефрагментации IP-пакетов и реассемблирования данных, содержащихся в TCP-сегментах, приведены в разделе А.9 приложения А (рисунки А.9 и А.10). В зависимости от сетевой нагрузки на сенсоре может быть запущено несколько анализаторов, каждый из которых прослушивает сетевой трафик на определенном интерфейсе, являющемся выходным для балансировщика. Серверная часть выполняет функции по приему данных от клиентских сенсоров, взаимодействие с которыми осуществляется при помощи механизма RPC с шифрованием SSL; для этого использовались компилятор rpcgen и deb-пакет libgnutls-openssl-dev. Адаптивные модули, представленные в виде динамических плагинов и построенные на основе методов ВИ, реализуют обучение соответствующих классификационных моделей и загружаются перед запуском детектора атак при помощи вспомогательных компонент (интерпретатора и менеджера классификаторов), входящих в состав интеллектуального ядра классификации объектов. В случае обнаруже-

ния атаки опционально осуществляется удаленная настройка правил файервола iptables на клиентских сенсорах при помощи команд, передаваемых по RPC/SSL.

Неотъемлемой частью в функционировании любой сетевой СОА является сетевой стек (подсистема обработки сетевых пакетов) с реализацией IP-дефрагментации и TCP-реассемблирования. Поскольку проведенное в разделе А.9 приложения А исследование открытых СОА показало ряд их существенных недостатков, связанных с обнаружением атак со скрытием и со вставкой, было принято решение отказаться от использования таких СОА в качестве сенсоров и разработать сетевой анализатор, способный корректно справляться со свойственными ОС Linux особенностями в сетях под управлением TCP/IP. Анализатор получил приставку „событийно-ориентированный“ благодаря встроенной в него функциональности вызова пользовательских функций при возникновении определенных событий, связанных с захваченными из сети пакетов. Всего таких событий было выделено 32, описание каждого из которых представлено в таблице 6 в порядке убывания очередности вызова (приоритета) сооответствующих им callback-функций. Разработанный анализатор состоит из двух компонент. Первый компонент, представленный в двух исполнениях, — это динамическая библиотека libnetpcap.so (статическая библиотека libnetpcap.a), построенная на основе библиотеки libpcap.so и предоставляющая весь необходимый API с поддержкой обработки сетевых событий и возможностью регистрации соответствующих им callback-функций. Второй компонент — это исполняемый бинарный ELF-файл netpcap_sensor, полученный посредством динамической линковки с библиотекой libnetpcap.so (статической линковки с библиотекой libnetpcap.a). Именно второй компонент занимается извлечением 106 параметров, характеризующих сетевые соединения; вычисление этих параметров выполняется в представленных callback-функциях. Удобство такого модульного решения заключается в том, что, когда возникает необходимость добавления или изменения каких-либо сетевых параметров, ядро анализатора, представленное библиотекой libnetpcap.so (libnetpcap.a), остается статичным, а повторной компиляции подвергается только исходный файл netpcap_sensor.

Таблица 6 — События анализатора сетевого трафика

№	Идентификатор события и callback-функция	Описание события
1	IP_FRAGMENT_INCORRECT_CHKSUM_EVENT, ip_fragment_incorrect_chksum_hndl	Событие появления IP-фрагмента (IP-пакета с установленным флагом MF или ненулевым смещением) с неправильной контрольной суммой заголовка
2	IP_FRAGMENT_ILLEGAL_FLAGS_EVENT, ip_fragment_illegal_flags_hndl	Событие появления IP-фрагмента с неправильными флагами (с единичным зарезервированным битом)
3	IP_ARRIVE_FRAGMENT_PACKET_EVENT, ip_arrive_fragment_packet_hndl	Событие появления IP-фрагмента
4	IP_OVERLAP_FRAGMENTS_EVENT, ip_overlap_fragments_hndl	Событие появления IP-фрагмента, частично или полностью перекрывающего ранее полученные IP-фрагменты с одинаковыми значениями IP-идентификаторов (наличие такого события может свидетельствовать о выполнении атак со скрытием/со вставкой согласно сценарию 4 или 6 раздела А.9 приложения А)
5	IP_INCORRECT_CHKSUM_EVENT, ip_incorrect_chksum_hndl	Событие появления IP-пакета с неправильной контрольной суммой заголовка
6	IP_ILLEGAL_FLAGS_EVENT, ip_illegal_flags_hndl	Событие появления IP-пакета с неправильными флагами (с единичным зарезервированным битом)
7	IP_ARRIVE_PACKET_EVENT, ip_arrive_packet_hndl	Событие появления IP-пакета
8	IP_REASSEMBLE_TIMEOUT_EVENT, ip_reassemble_timeout_hndl	Событие удаления просроченных IP-фрагментов (событие возникает в двух случаях: (1) при условии необновления списка IP-фрагментов в течение заданного промежутка времени либо (2) по достижении специфичного для каждой ОС системного таймаута, выделенного на сборку IP-фрагментов и приводящего к генерации ICMP-пакета с типом 11 и кодом 1 стороной A, производящей IP-дефрагментацию; во втором случае наличие таких временных задержек может указывать на сканирование на уровне IP стороной B, которая присылает фрагментированные пакеты, — этот трюк часто используется стороной B для определения версии ОС стороны A либо для проведения атак со скрытием/со вставкой согласно сценарию 7 раздела А.9 приложения А)
9	TCP_INCORRECT_CHKSUM_EVENT, tcp_incorrect_chksum_hndl	Событие появления TCP-сегмента с неправильной контрольной суммой псевдозаголовка

Продолжение таблицы 6

№	Идентификатор события и callback-функция	Описание события
10	TCP_ILLEGAL_FLAGS_EVENT, tcp_illegal_flags_hndl	Событие появления TCP-пакета с неправильными/конфликтующими флагами (как правило, появление таких пакетов является индикатором сканирования для проверки доступности TCP-порта)
11	TCP_ARRIVE_PACKET_EVENT, tcp_arrive_packet_hndl	Событие появления TCP-пакета
12	TCP_RETRANSMIT_PACKET_EVENT, tcp_retransmit_packet_hndl	Событие появления TCP-пакета с позиционным номером, повторяющимся с позиционным номером одного из ранее полученных (отправленных) TCP-сегментов или наслаивающимся с диапазоном позиционных номеров других TCP-сегментов в пределах текущего TCP-окна (как правило, появление таких пакетов в сети вызвано тем, что (1) принимающая такие пакеты сторона B задерживает отправку пакетов, направленных адресату A с подтверждением в их получении, и тем самым сторона B пытается выполнить атаку сканирования по таймауту для определения настроек сетевой TCP-службы A или (2) отправляющая такие пакеты сторона пытается провести атаку со скрытием/со вставкой согласно сценарию 13 или 15 раздела А.9 приложения А)
13	TCP_IS_ABSENT_SYN_EVENT, tcp_is_absent_syn_hndl	Событие появления TCP-пакета с неустановленным флагом SYN для несуществующей сессии (как правило, появление таких пакетов свидетельствует о сканировании сетевых служб на уровне TCP либо о пропуске момента синхронизации позиционных номеров клиента и сервера при попытке открытия сессии)
14	TCP_START_SESSION_EVENT, tcp_start_session_hndl	Событие появления TCP-пакета с установленным флагом SYN для несуществующей сессии (событие попытки для открытия новой TCP-сессии)
15	TCP_ESTABLISH_3HANDSHAKE_EVENT, tcp_establish_3handshake_hndl	Событие появления последнего пакета, приводящего к установлению TCP-соединения с двусторонней синхронизацией позиционных номеров (событие „троекратного рукопожатия“; после возникновения этого события осуществляется реальный обмен данными)
16	TCP_TRANSMIT_DATA_EVENT, tcp_transmit_data_hndl	Событие появления пакета после выполнения процесса полного установления TCP-соединения

Продолжение таблицы 6

№	Идентификатор события и callback-функция	Описание события
17	TCP_RESET_SESSION_EVENT, tcp_reset_session_hndl	Событие сброса TCP-соединения посредством отправки одной из сторон TCP-пакета с установленными флагом RST
18	TCP_CLOSE_SESSION_EVENT, tcp_close_session_hndl	Событие закрытия TCP-соединения (событие успешного перехода в терминальное состояние TCP-сессии CLOSED)
19	TCP_EXCEED_EXPIRE_TIMEOUT_EVENT, tcp_exceed_expire_timeout_hndl	Событие удаления TCP-сессии по причине (1) ее закрытия, (2) ее устарелости или (3) нехватки системных ресурсов для добавления новой TCP-сессии
20	TCP_REASSEMBLE_TIMEOUT_EVENT, tcp_reassemble_timeout_hndl	Событие удаления реассемблируемых TCP-сегментов по причине их необновления в течение заданного таймаута
21	UDP_INCORRECT_CHKSUM_EVENT, udp_incorrect_chksum_hndl	Событие появления UDP-пакета с неправильной контрольной суммой псевдозаголовка
22	UDP_EMPTY_PACKET_EVENT, udp_empty_packet_hndl	Событие появления UDP-пакета с пустым содержанием (наличие таких пакетов может указывать на сканирование на уровне UDP)
23	UDP_ARRIVE_PACKET_EVENT, udp_arrive_packet_hndl	Событие появления UDP-пакета
24	UDP_START_SESSION_EVENT, udp_start_session_hndl	Событие открытия UDP-потока (событие регистрации первого UDP-пакета между парой сокетов при условии отсутсвия более ранних пакетов в рамках заданного интервала времени)
25	UDP_EXCEED_EXPIRE_TIMEOUT_EVENT, udp_exceed_expire_timeout_hndl	Событие закрытия UDP-потока по причине необновления UDP-пакетов в течение заданного промежутка времени
26	ICMP_INCORRECT_CHKSUM_EVENT, icmp_incorrect_chksum_hndl	Событие появления ICMP-пакета с неправильной контрольной суммой псевдозаголовка
27	ICMP_EMPTY_REQUEST_EVENT, icmp_empty_request_hndl	Событие появления ICMP-запроса с пустым содержанием (наличие таких пакетов может указывать на сканирование на уровне ICMP)
28	ICMP_ECHO_REQUEST_EVENT, icmp_echo_request_hndl	Событие появления ICMP-запроса
29	ICMP_ECHO_REPLY_EVENT, icmp_echo_reply_hndl	Событие появления ICMP-ответа
30	ICMP_ARRIVE_PACKET_EVENT, icmp_arrive_packet_hndl	Событие появления ICMP-пакета
31	ICMP_START_SESSION_EVENT, icmp_start_session_hndl	Событие открытия ICMP-потока (запрос-ответ)
32	ICMP_EXCEED_EXPIRE_TIMEOUT_EVENT, icmp_exceed_expire_timeout_hndl	Событие закрытия ICMP-потока (по причине его устарелости)

Прототип callback-функций представляется следующим образом:

```
typedef void (*proto_handler)
    (const ip_connection *ip_conn,
     u_char *user_data,
     const void *pkt,
     uint16_t pkt_len,
     uint8_t from_client,
     const struct timeval *timestamp);
```

Первый параметр во всех callback-функциях — это указатель на объект (типа ip_connection), содержащий данные, связанные с потоком IP-пакетов. Второй аргумент представляет собой указатель на буфер пользовательских данных, специфичных для каждого конкретного протокола уровня не ниже IP. В третьем и четвертом аргументах передаются соответственно указатель на заголовок пакета и размер реально захваченных в нем данных (вместе с размером заголовка). Пятый аргумент является бинарным флагом, указывающим на направление этого пакета (from_client принимает значение 1, если пакет поступил от инициатора сессии, и 0 в противном случае). В последнем аргументе содержится значение метки времени захвата пакета на сетевом интерфейсе. Добавление callback-функции hndl, удовлетворяющей данному прототипу и предназначенной для анализа пакетов на уровне одного из четырех протоколов proto=IP|TCP|UDP|ICMP, в список обработчиков события event выполняется при помощи вызова функции netpcap_register_proto_callback с фактическими параметрами hndl, proto, event:

```
size_t netpcap_register_proto_callback
    (proto_handler hndl, uint8_t proto, uint8_t event);
```

Удаление обработчика события event, свойственного для протокола proto, осуществляется при помощи вызова функции netpcap_unregister_proto_callback с фактическими параметрами proto, event и id (идентификатора, возвращенного функцией netpcap_register_proto_callback):

```
void netpcap_unregister_proto_callback
    (uint8_t proto, uint8_t event, size_t id);
```

Структура для хранения IP-потока представляется следующим образом:

```
#pragma pack(push)
#pragma pack(1)
typedef struct {
    uint32_t src_addr; // in network order
    union {
        uint16_t src_port; // in host order (TCP and UDP)
        uint16_t icmp_echo_id; // in host order (ICMP)
    };
    uint32_t dst_addr; // in network order
    union {
        uint16_t dst_port; // in host order (TCP and UDP)
        uint16_t icmp_echo_seq; // in host order (ICMP)
    };
    uint8_t ip_proto; // TCP (6), UDP (17), ICMP (1)
} socket_pair;
typedef struct ip_connection {
    socket_pair sock_pair;
    struct timeval sndr_first_pkt_timestamp;
    struct timeval sndr_last_pkt_timestamp;
    struct timeval rcvr_first_pkt_timestamp;
    struct timeval rcvr_last_pkt_timestamp;
    uint32_t hash_value;
    union {
        tcp_connection tcp_conn;
        udp_connection udp_conn;
        icmp_connection icmp_conn;
    };
    u_char *user_data_l4; // user data (TCP, UDP, ICMP)
    u_char *user_data_l3; // user data (IP)
    uint8_t ttl; // time to live in the first packet
    uint8_t tos; // type of service in the first packet
    struct timeval first_pkt_timestamp;
    struct timeval last_pkt_timestamp;
    struct ip_connection *next_node;
    struct ip_connection *prev_node;
    struct ip_connection *next_free;
    struct ip_connection *older_conn;
    struct ip_connection *newer_conn;
    scan_active_connection *s_active_conns; // scan
    struct ip_connection *older_s_conn;
    struct ip_connection *newer_s_conn;
    dos_active_connection *d_active_conns; // DoS
    struct ip_connection *older_d_conn;
    struct ip_connection *newer_d_conn;
} ip_connection;
#pragma pack(pop)
```

В алгоритме 22 представлены функции преаллокации и освобождения памяти (init_ip_memory и free_ip_memory), добавления, удаления и поиска IP-потока (add_new_ip_connection, remove_ip_connection и find_ip_connection).

Алгоритм 22: Алгоритм преаллокации памяти в сетевом анализаторе

```
1  static ip_connection *ip_conn_pool := *free_ip_conns := *last_added_conn := **ip_conn_table := NULL
2  static uint32_t cur_ip_count := 0
3  void init_ip_memory()
4  начало блока
5      ip_conn_table := calloc(MAX_IP_TABLE_SIZE,sizeof(ip_connection *))
6      ip_conn_pool := malloc(MAX_IP_POOL_SIZE * sizeof(ip_connection))
7      цикл i от 0 до MAX_IP_POOL_SIZE − 1 выполнять
8          ip_conn_pool[i].next_free := (i < MAX_IP_POOL_SIZE − 1?ip_conn_pool + i + 1 : NULL)
9          ip_conn_pool[i].user_data_l4 := malloc(USER_DATA_L4_LEN)
10         ip_conn_pool[i].user_data_l3 := malloc(USER_DATA_L3_LEN)
11     free_ip_conns = ip_conn_pool
12 void free_ip_memory()
13 начало блока
14     освободить память, выделенную под ip_conn_table и ip_conn_pool
15 ip_connection *add_new_ip_connection(const socket_pair *sock_pair, const struct timeval *timestamp)
16 начало блока
17     если cur_ip_count >= MAX_IP_POOL_SIZE тогда
18         удалить устаревшие IP-потоки из ip_conn_table
19         если cur_ip_count >= MAX_IP_POOL_SIZE тогда
20             возвратить NULL
21     ip_connection *new_ip_conn := free_ip_conns
22     free_ip_conns := free_ip_conns→next_free
23     ++cur_ip_count
24     hash_key := calc_fnv_hash_key(sock_pair)
25     new_ip_conn→hash_value := hash_key
26     memset(new_ip_conn→user_data_l4, 0, USER_DATA_L4_LEN)
27     memset(new_ip_conn→user_data_l3, 0, USER_DATA_L3_LEN)
28     new_ip_conn→first_pkt_timestamp := new_ip_conn→sndr_first_pkt_timestamp := *timestamp
29     new_ip_conn→sock_pair := *sock_pair
30     new_ip_conn→next_node := ip_conn_table[hash_key]
31     new_ip_conn→prev_node := NULL
32     если ip_conn_table[hash_key]! = NULL тогда
33         ip_conn_table[hash_key]→prev_node := new_ip_conn
34     new_ip_conn→older_conn := last_added_conn
35     new_ip_conn→newer_conn := NULL
36     если last_added_conn! = NULL тогда
37         last_added_conn→newer_conn := new_ip_conn
38     last_added_conn := new_ip_conn
39     ip_conn_table[hash_index] := new_ip_conn
40     возвратить new_ip_conn
41 void remove_ip_connection(ip_connection *ip_conn)
42 начало блока
43     если ip_conn→next_node! = NULL тогда
44         ip_conn→next_node→prev_node := ip_conn→prev_node
45     если ip_conn→prev_node! = NULL тогда
46         ip_conn→prev_node→next_node := ip_conn→next_node
47     иначе
48         ip_conn_table[ip_conn→hash_value] := ip_conn→next_node
49     если ip_conn→older_conn! = NULL тогда
50         ip_conn→older_conn→newer_conn := ip_conn→newer_conn
51     если ip_conn→newer_conn! = NULL тогда
52         ip_conn→newer_con→older_conn := ip_conn→older_conn;
53     иначе
54         last_added_conn := ip_conn→older_conn
55     ip_conn→next_free := free_ip_conns
56     free_ip_conns := ip_conn
57     −−cur_ip_count
58 ip_connection *find_ip_connection(const socket_pair *sock_pair)
59 начало блока
60     ip_connection *ip_c := ip_conn_table[calc_fnv_hash_key(sock_pair)]
61     пока ip_c! = NULL&&memcmp(&ip_c→sock_pair, sock_pair, sizeof(socket_pair))! = 0 выполнять
62         ip_c := ip_c→next_node
63     если ip_c = NULL тогда
64         поменять местами сокеты в sock_pair и повторить вышеописанный цикл
65     возвратить ip_conn
```

Механизм хранения хешированных записей, задающих IP-потоки, во многом схож с подходом, используемым в пакетном балансировщике из раздела A.10 приложения A. Здесь используются два глобальных объекта — free_ip_conns и ip_conn_table, инициализация которых выполняется в функции init_ip_memory. Первый из них представляет собой указатель на головной элемент списка, содержащий объекты типа ip_connection, для которых предварительно выделена память при помощи операции malloc с целью их дальнейшего размещения в объекте ip_conn_table. Второй объект является массивом (указателем на указатели объектов типа ip_connection), в котором доступ к каждой записи IP-потока организуется при помощи индекса, вычисленного как хеш-функция fnv от объекта типа socket_pair. Для каждого захваченного пакета с сокетной парой sock_pair проверяется его факт принадлежности массиву ip_conn_table при помощи функции find_ip_connection. Если результат возврата этой функции равен NULL, то в массив ip_conn_table добавляется новый элемент при помощи функции add_new_ip_connection: сперва указатель free_ip_conns перемещается на соседний элемент одноименного списка при помощи ссылки next_free, а затем свежеизвлеченный из начала этого списка элемент добавляется в роли головного в другой список ip_conn_table[calc_fnv_hash_key(sock_pair)], инициализируется и возвращается как выходное значение функции add_new_ip_connection. Если же результат возврата функции find_ip_connection отличен от NULL, то полученный пакет помечается как принадлежащий существующему IP-потоку, который должен быть обновлен в соответствии с полями захваченного пакета. Для удаления IP-потока используется процедура remove_ip_connection, обратная по действиям функции add_new_ip_connection и перенаправляющая указатели таким образом, чтобы данный IP-поток был исключен из массива ip_conn_table и вновь размещен в списке free_ip_conns. Использование такого приема с предварительной аллокацией памяти на старте программы позволяет сократить время ее выполнения и избежать лишних вызовов функции malloc непосредственно уже в процессе анализа сетевого пакета и обработки IP-потока. Освобождение ранее выделенных динамических ресурсов выполняется в функции free_ip_memory.

Кроме того, данный подход с использованием таблицы ip_conn_table и функции хеширования fnv позволяет сократить количество операций, связанных с линейным поиском интересующего IP-потока с сокетной парой sock_pair: такой поиск выполняется лишь только внутри определенной части таблицы, соответствующей ip_conn_table[calc_fnv_hash_key(sock_pair)]. Для обеспечения быстрого доступа к соединениям, которые имеют одинаковый с данным IP-потоком IP-адрес источника (назначения), в структуру ip_connection были добавлены перекрестные ссылки s_active_conns, older_s_conn, newer_s_conn (d_active_conns, older_d_conn, newer_d_conn). Эти поля содержат указатели на такие соединения, отсортированные по времени их добавления в таблицу ip_conn_table, а также дополнительную информацию о пакетах, характеризующих атаки типа сканирования хостов (отказа в обслуживании).

Реализованная функция calc_fnv_hash_key для вычисления хеш-ключа fnv была изменена таким образом, чтобы для любой пары сокетов в рамках IP-потока ее значение было всегда одинаковым вне зависимости от того, кому был направлен пакет (клиенту или серверу). Для этого передаваемый в нее аргумент сперва сортируется по сокетам отправителя и получателя, и только затем к полученному 13-байтовому значению применяется хеш-функция. Итоговый результат будет инвариантным по отношению к любому пакету внутри определенного IP-потока.

Для описания гибридных схем, задающих объединение разнообразных бинарных классификаторов в единый высокоуровневый классификатор, был реализован интерпретатор интеллектуального ядра классификации объектов, формальная грамматика которого представлена в приложении В. При его разработке использовались GNU-инструменты flex и bison, предназначенные для генерации синтаксических таблиц типа LALR(1). Результатом работы интерпертатора над входным файлом является объект древовидной структуры, в котором каждым узлом является классификатор, — дерево классификаторов. Синтаксис обрабатываемых этим интерпретатором конфигурационных файлов является JSON-подобным, и ниже представлен сокращенный пример содержимого такого файла.

```
classifier_tree: {
  vars: {                                                    ┐
    in_dim: {                                                │
      "33" }                                                 │ блок объявления констант
    out_dim: {                                               │
      "6" } }                                                ┘
  classifier: {
    name: {                                                  ┐ название корневого
      "meta_neural_network_classifier" }                     ┘ классификатора
    identifier: {                                            ┐ идентификатор корневого
      "7" }                                                  ┘ классификатора
    so_library: {                                            ┐ подключаемый плагин
      "libmeta_neural_network_classifier.so" }               ┘ корневого классификатора
    prf_file: {                                              ┐ название файла с
      "meta_classifier_results.prf" }                        ┘ выходными результатами
    sls_file: {                                              ┐ файл сериализации
      "meta_classifier_structure.sls" }                      ┘ структуры классификатора
    opt_file: {                                              ┐ файл с опциями
      "meta_classifier_options.opt" }                        ┘ классификатора
    input_dim: {                                             ┐ размерность
      "7" }                                                  ┘ входного вектора
    output_dim: {                                            ┐ размерность
      "$out_dim" }                                           ┘ выходного вектора
    input_data: {                                            ┐ дерево выражений,
      "concat(sum(idx#4,idx#5,idx#6),idx#3)" }               ┘ аргументы которого - список зависимостей корневого классификатора
    output_data: {
      type: {                                                ┐ тип выходных
        "array" } }                                          ┘ данных (массив)
    nested_classifiers: {                                    ┐ дочерние узлы
      classifier: {
        name: {                                              ┐ название узлового
          "neural_network_classifier" }                      ┘ классификатора
        identifier: {                                        ┐ идентификатор узлового
          "6" }                                              ┘ классификатора
        so_library: {                                        ┐ подключаемый плагин
          "libneural_network_classifier.so" }                ┘ узлового классификатора
        ...
        multithread: {                                       ┐ режим многопоточности
          "on" }                                             ┘
        bagging: {                                           ┐ механизм бэггинга
          "on" }                                             ┘
        num_groups: {                                        ┐ количество групп
          "3" }                                              ┘
        input_dim: {                                         ┐ размерность
          "$in_dim" }                                        ┘ входного вектора
        output_dim: {                                        ┐ размерность
          "$out_dim" }                                       ┘ выходного вектора
        input_data: {                                        ┐ дерево выражений,
          "idx#2" }                                          ┘ аргументы которого - список зависимостей узлового классификатора
        output_data: {
          type: {                                            ┐ тип выходных
            "array" }                                        ┘ данных (массив)
          format: {                                          ┐
            detector: {                                      │
              ...                                            │ настройки и правила
              rules: { ... } }                               │ обучения вложенных
            ...                                              │ в узловой классификатор
            detector: {                                      │ детекторов
              ...                                            │
              rules: { ... } } } }                           ┘
        nested_classifiers: { ... } }
      ...
      classifier { ... } } } }
```

Менеджер классификаторов выполняет следующие задачи: (1) загрузка классификаторов из so-библиотек (плагинов), (2) вызов функций, определенных в плагинах, (3) иерархический обход дерева классификаторов. Первая и вторая задачи решаются при помощи системных функций dlopen (или dlmopen) и dlsym. В результате вызова первой функции возвращается дескриптор открываемого плагина, применение к которому второй функции позволяет получить доступ непосредственно к указателю требуемого объекта (функции или переменной) по его имени внутри плагина. Вызовы этих функций осуществляются в процессе выполнения программы, поэтому линковку плагинов с ядром классификации выполнять не требуется. Можно сказать, что менеджеру классификаторов ничего неизвестно ни о внутренней структуре каждого из классификаторов, ни об алгоритмах его обучения и формирования выходного вектора. Все управление классификаторами сводится к вызову одной из семи функций, которые должны быть обязательно определены в каждом плагине: (1) create — функция создания классификатора; (2) train — функция обучения классификатора; (3) save — функция сохранения структуры (сериализации) классификатора в файл; (4) load — функция загрузки структуры (десериализации) классификатора из файла; (5) run — функция тестирования классификатора без априорных сведений о классе входного объекта; (6) examine — функция тестирования классификатора с учетом наличия сведений о классе входного объекта; (7) destroy — функция уничтожения классификатора. Тем самым ядро классификации и загружаемые в него плагины могут разрабатываться независимо друг от друга. Среди некоторых других ограничений, накладываемых на плагины, стоит отметить, что при их разработке рекомендуется отказаться от использования глобальных переменных, поскольку повторная загрузка библиотеки при помощи dlopen возвращает ранее связанный с ней дескриптор. В этом случае две, как казалось бы независимые, копии одного классификатора будут на самом деле иметь доступ к одному и тому же глобальному (или статическому) объекту. Незнание этого факта может приводить к неочевидным ошибкам, которые трудно обнаружить в процессе выполнения программы. Если использования глобальных объектов не удается избежать,

то необходимо исключить случаи взаимного влияния нескольких копий одного и того же классификатора при получении доступа к этим объектам. Кроме того, на некоторых платформах эту проблему можно решить при помощи функции dlmopen, которая создает при каждом ее вызове новое отдельное адресное пространство для размещения в нем всех объектов загружаемой библиотеки. Закрытие дескриптора ранее открытой библиотеки осуществляется при помощи вызова функции dlclose. За счет использования такого подхода, связанного с разбиением системы на обрабатывающие компоненты (плагины) и управляющий ими компонент (менеджер классификаторов), появляется возможность горячей вставки нового исполняемого кода „на лету“ (без останова системы). Для подтверждения этого факта сперва необходимо изменить надлежащим образом конфигурационный файл с описаниями добавляемых классификаторов и запустить интерпретатор для построения дерева классификаторов (к примеру, в обработчике сигнала SIGUSR1), после чего менеджер классификаторов без перезагрузки способен выполнять анализ входных векторов согласно новой конфигурации.

Было реализовано 20 плагинов, предназначенных для загрузки в интеллектуальное ядро классификации объектов и представленных в таблице 7. За счет такого разнообразия применяемых плагинов интеллектуальное ядро классификации объектов поддерживает все описанные в главе 2 методы вычислительного интеллекта с использованием большого количества настроек и алгоритмов обучения соответствующих классификационных моделей. Бóльшая часть из вспомогательных библиотек, использующихся на нижнем уровне и лежащих в основе плагинов, реализована лично автором настоящего диссертационного исследования. При разработке плагинов использовалось 5 АЯВУ: C, C++, Perl, Python, R. Для возможности вызова функций Perl-, Python- и R-скриптов из кода, написанного на C, были реализованы вспомогательные библиотеки, осуществляющие маршалинг различных типов и структур данных (строк, чисел и списков/массивов) во внутренний формат данных, воспринимаемый компилятором gcc языка программирования C. Основой для этих библиотек послужили библиотека C libperl.so, обертка библиотеки C++ libboost_python.so и модуль Python rpy2.

Таблица 7 — Плагины интеллектуального ядра классификации объектов

№	Плагин и его описание	Используемые в плагине библиотеки, скрипты и модули
1	libcompetence_referee.so, арбитр на основе динамических областей компетентности (метод Фикса–Ходжеса)	libc.so (C)
2	libdim_reducer.so, метод главных компонент (МГК)	libpython_func_wrapper.so, libpython_ffi.so, libboost_python.so (C++); libpython2.7.so, libperl_ffi.so, libperl.so, libdl.so, libm.so, libpthread.so, libcrypt.so, libc.so (C); pca.py, collections, cPickle, sklearn, matplotlib, numpy (Python); pca.pl, Statistics::PCA, Data::Dumper, Math::Cephes::Matrix, Math::MatrixReal, Storable (Perl)
3	libevolutional_immune_system.so, искусственная иммунная система на базе эволюционного подхода	libimmune_system.so, libkohonen_card.so, libgenetic_recombinator.so, libboost_system, libboost_thread, libboost_chrono (C++); libm.so, libc.so (C)
4	libfalse_classifier.so, ложный классификатор	libc.so (C)
5	libfield_extractor.so, слайсинг заданных полей из входного вектора	libc.so (C)
6	libidentical_mapper.so, генератор бинарного массива	libc.so (C)
7	libkmeans_clusterization.so, метод K-средних	libkmeans.so, libm.so, libc.so (C)
8	libknn_clusterization.so, метод ближайших соседей	libknn.so, libm.so, libc.so (C)
9	libkohonen_card_classifier.so, классификатор на основе карт Кохонена	libkohonen_card.so, libgenetic_recombinator.so, libboost_system (C++); libm.so, libpthread.so, libc.so (C)
10	libmajority_voter.so, метод мажоритарного голосования	libc.so (C)
11	libmeta_neural_network_classifier.so, нейросетевой мета-классификатор	libfann.so, libc.so (C)
12	libmixed_neural_network_classifier.so, „смешанный“ классификатор на основе трех типов нейронных сетей (с 3 типами функций активации и 16 алгоритмами обучения), машины опорных векторов (с 4 типами ядер) и нейронечеткой сети типа ANFIS (с 3 типами функций принадлежности и 3 алгоритмами обучения)	libpython_func_wrapper.so, libpython_ffi.so, libboost_python.so (C++); libpython2.7.so, libc.so (C); neupy_ff_neural_network.py, peach_rbf_neural_network.py, pybrain_rec_neural_network.py, sklearn_svm.py, anfis_neurofuzzy_network.py, rpy2_anfis_neurofuzzy_network.py, neupy, peach, pybrain, sklearn, numpy, dill, logging, cPickle, rpy2, anfis (Python); r_anfis.R, anfis, methods, parallel, hash, pryr (R)
13	libneural_network_classifier.so, нейросетевой классификатор с 3 типами функций активации и 3 алгоритмами обучения	libfann.so, libm.so, libpthread.so, libc.so (C)
14	libneurofuzzy_classifier.so, нейронечеткий классификатор типа ANFIS	libanfis.so, libgsl.so, libgslcblas.so, libm.so, libpthread.so, libc.so (C)
15	libparameter_normalizer.so, метод нормализации параметров	libc.so (C)
16	libpca_reducer.so, МГК	libpca.so, libgsl.so, libgslcblas.so, libc.so (C)
17	librandom_classifier.so, случайный классификатор	libc.so (C)
18	libsimple_neural_network_classifier.so, нейросетевой классификатор с 3 типами функций активации (сети прямого распространения сигнала, сети с РБФ, рекуррентные сети Джордана и Элмана)	libsimple_neural_network.so, libm.so, libpthread.so, libc.so (C)
19	libsvm_classifier.so, классификатор на основе машины опорных векторов (с 4 типами ядерных функций)	libsvmlight.so, libm.so, libpthread.so, libc.so (C)
20	libweight_voter.so, метод взвешенного голосования	libc.so (C)

Иерархический обход дерева классификаторов, третья задача менеджера классификаторов, заключается в посещении каждого из узлов дерева классификаторов в заданном порядке с целью вызова одной или нескольких функций, определенных в плагинах. Наиболее интересным случаем является обучение узловых классификаторов: необходимо чередовать процедуры формирования входных векторов и обучения на этих данных, т.е. вызовы функций examine и train. Алгоритм этого процесса, называемого каскадным обучением дерева классификаторов, приведен в алгоритме 23. Сперва выполняется обход в ширину дерева классификаторов по ссылкам на дочерние узлы, затем осуществляется каскадное обучение узла и всех его зависимостей согласно его дереву выражений.

Алгоритм 23: Алгоритм каскадного обучения дерева классификаторов

```
1  функция f_1 конвертирования дерева классификаторов cls_tree в однонаправленный список dir_list
2  начало блока
3      dir_list :=создать пустой список
4      добавить корень root_node дерева cls_tree в качестве головного элемента в список dir_list
5      list_elem :=получить головной элемент списка dir_list (root_node)
6      пока list_elem не равен NULL выполнять
7          если узел list_elem не является терминальным в дереве cls_tree тогда
8              для каждого дочернего узла nested_node элемента list_elem выполнять
9                  добавить узел nested_node в качестве хвостового элемента в список dir_list
10         list_elem :=получить следующий за list_elem элемент из списка dir_list
11     возвратить dir_list
12 функция f_2 каскадного обучения узла node и его зависимостей в дереве классификаторов cls_tree
13 начало блока
14     если узел node помечен как необученный тогда
15         если узел node нетерминальный тогда
16             для каждого узла dep_node из списка зависимостей узла node выполнять
17                 вызвать функцию f_2 для узла dep_node
18         in_data :=сформировать входные векторы для узла node согласно его дереву выражений
19         выполнить обучение классификатора, размещенного в узле node, на данных in_data
20         пометить узел node как обученный
21 функция f_3 обхода дерева классификаторов cls_tree в ширину при помощи списка dir_list
22 начало блока
23     node :=получить головной элемент списка dir_list
24     пока node не равен NULL выполнять
25         вызвать функцию f_2 для узла node
26         node :=получить следующий за node элемент из списка dir_list
27 cls_tree :=вызвать интерпретатор для заданного пользователем конфигурационного файла
28 dir_list :=вызвать функцию f_1 для дерева классификаторов cls_tree
29 вызвать функцию f_3 для дерева классификаторов cls_tree и списка dir_list
```

На данный момент операции чтения и записи сетевых параметров осуществляются согласно трем форматам хранения и представления данных: обычные текстовые CSV-файлы, реляционные таблицы БД MySQL, коллекции NoSQL БД MongoDB.

Как уже говорилось ранее, для обмена данными между сенсорами и коллектором используется туннелированный внутри SSL протокол RPC. В некоторых случаях разумным решением является предварительная буферизация вычисляемых сетевых параметров, что позволит переложить обязанности по их дальнейшей передаче на другие средства, отличные от RPC/SSL. Поэтому для обеспечения гибкой интеграции сенсоров с внешними системами, например Weka, RapidMiner, выполняющими схожие с интеллектуальным ядром классификации объектов функции, в состав netpcap_sensor включен ряд дополнительных опций, предназначенных для генерации выходных CSV-файлов. В них сформированный список сетевых параметров представляется как отдельная текстовая строка. Кроме того, в программе netpcap_sensor предоставляется возможность генерации CSV-файлов с заранее заданной пользователем частотой (ротацией), которая выполняется при достижении одного из условий: превышения заданного таймаута, превышения заданного размера файла или превышения заданного числа записей в нем; после чего начинается формирование очередных полей с записью в свежесозданный файл. Одна из опций разработанной программы позволяет определить команду, которая будет вызываться всякий раз в фоновом режиме (как отдельный процесс) после того, как CSV-файл полностью сформирован. Предполагается, что одной из таких команд будет являться команда (к примеру, scp или rsync), обеспечивающая копирование сгенерированного файла на удаленный сервер, где будет осуществляться анализ и интерпретация полученных данных при помощи какой-либо из вышепредставленных систем.

3.3 Архитектура и программная реализация стенда генерации сетевых атак

Разработанный стенд генерации сетевых атак состоит из двух независимых компонент:

– Асинхронный прозрачный прокси-сервер TCP-сессий;

– Генератор сетевых атак и frontend-интерфейс к нему.

Каждый из названных компонент нацелен на генерацию разнообразных сетевых атак, которые могут включать возможные способы выполнения уклонений от сигнатурных сетевых СОА на различных уровнях модели OSI и представлять собой аномальную сетевую активность, которая характеризуется сканированием хостов или приводит к отказу в обслуживании хоста. Первый компонент предназначен для тестирования сигнатурных сетевых СОА с целью проверки их способности к обнаружению атак со скрытием и со вставкой. Подробное описание этого программного средства (включая его программную архитектуру, состав, алгоритм функционирования и примеры генерируемых с помощью него сетевых атак) приведено в разделе А.9 приложения А. В данном разделе описывается функционирование, архитектура и состав именно второго компонента.

Генератор сетевых атак позволяет выполнять определенные пользователем команды, которые направлены на формирование аномальных сетевых пакетов и их потоков (сетевого трафика) с целью дальнейшей их отправки на IP-адрес жертвы и/или сохранения в файл для дальнейшего детального изучения.

Для обеспечения отказоустойчивости данного генератора атак была разработана специальная архитектура, которая позволяет выполнять независимо сразу несколько процессов, осуществляющих те или иные операции с сетевыми пакетами (формирование полей и контента пакетов, перехват пакетов, запись захваченных пакетов с сохранением в pcap-файл, отправка пакетов, фильтрация пакетов, генерация аномальных пакетов и сессий, подмена полей в пакетах, перенаправление трафика в сетевой интерфейс из другого интерфейса или файла). На рисунке 3.8 изображена архитектура этого программного средства.

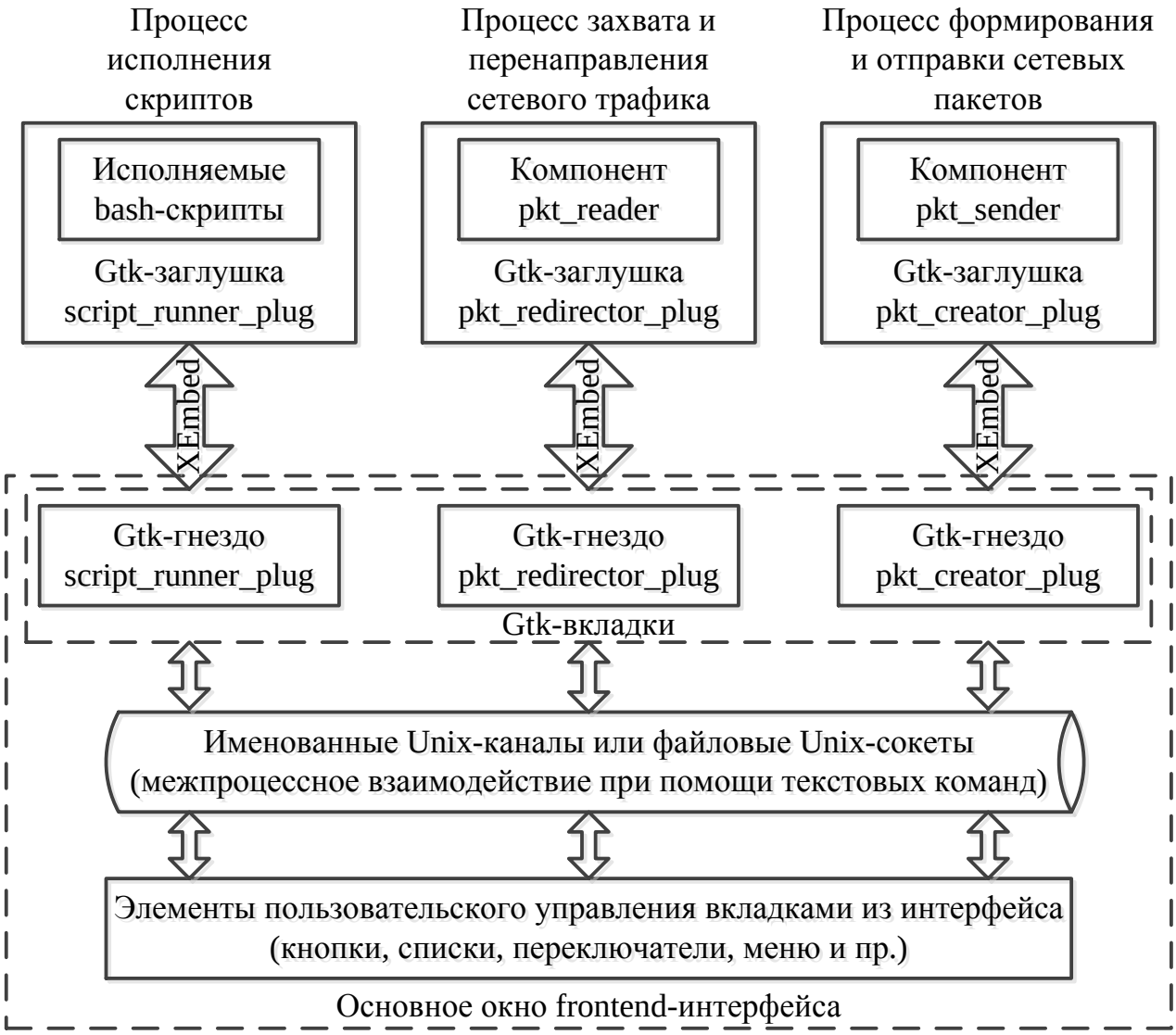


Рисунок 3.8 — Архитектура генератора сетевых атак

Генератор сетевых атак состоит из трех компонент: набор исполняемых скриптов, компонент pkt_reader, компонент pkt_sender. Описание последних двух компонент представлено в разделе А.9 приложения А. Представленные компоненты — это бинарные и скриптовые исполняемые файлы, которые имеют множество входных параметров и запускаются из Linux-консоли. Для предоставления более удобного доступа к основным функциям этих компонент был разработан frontend-интерфейс, внутри которого каждый компонент может быть запущен как самостоятельный процесс в отдельной Gtk-заглушке причем в нескольких копиях. Преимущество такого подхода заключается в том, что даже после аварийного завершения соответствующего процесса по причине наличия в его коде критической ошибки, к примеру ошибки сегментации (segmentation fault),

основное окно приложения и другие ранее запущенные в рамках него процессы останутся работоспособными, и как следствие генерация сетевого трафика и сетевых атак не прервется в неожиданном месте. В то же время порождение новых процессов приводит к увеличению объема потребляемой памяти, однако для современной вычислительной техники, обладающей большими аппаратными ресурсами, создание нескольких дополнительных процессов не сказывается существенно отрицательно на производительности системы. Встраивание обернутых в Gtk-заглушки процессов осуществляется при помощи созданных в основном окне приложения Gtk-гнезд, взаимодействие которых с Gtk-заглушками на системном уровне осуществляется при помощи протокола XEmbed. Сами Gtk-гнезда размещаются в основном окне приложения как вкладки виджета Gtk::Notebook, который позволяет добавлять новые созданные таким образом процессы или переключаться между уже запущенными процессами. Поскольку каждая вкладка и основное окно представляют собой разные процессы, то необходимо предусмотреть механизм их межпроцессного взаимодействия [33]. В данном программном средстве в роли такой связки задействуются именованные Unix-каналы и файловые Unix-сокеты, инициализируемые через системные вызовы mkfifo и socket (конкретный тип и имя файла определяются в качестве входных опций frontend-интерфейса). Для асинхронного отслеживания состояния открытых дескрипторов каналов и сокетов используется последовательность Glib-библиотечных вызовов g_io_channel_unix_new и g_io_add_watch [139], которая позволяет добавить callback-функцию и считывать в ней данные из дескрипторов в основном цикле обработки сигналов, аналогично вызову слотов. Эта функция вызывается только при появлении в канале или сокете новых еще не прочитанных данных. Конкретный состав вкладок формируется динамически из меню и указывается как входной аргумент frontend-интерфейса. За счет этого достигается важная особенность программного средства: добавление, изменение или удаление какой-либо Gtk-заглушки не влияет на исходный код frontend-интерфейса, и тем самым эти компоненты могут разрабатываться как самостоятельные модули. На данный момент поддерживаются три типа вкладок:

1. ScriptRunnerPlug (рисунок 3.9). Данная вкладка предназначена для запуска бинарных и скриптовых исполняемых файлов, генерирующих аномальный и/или нормальный трафик. Основные опции включают IP-адрес и порт отправителя, IP-адрес и порт получателя. Перехватываемый выход (потоки stdout и stderr) каждой из запускаемых команд отображается в текстовом буфере в нижней части интерфейса;

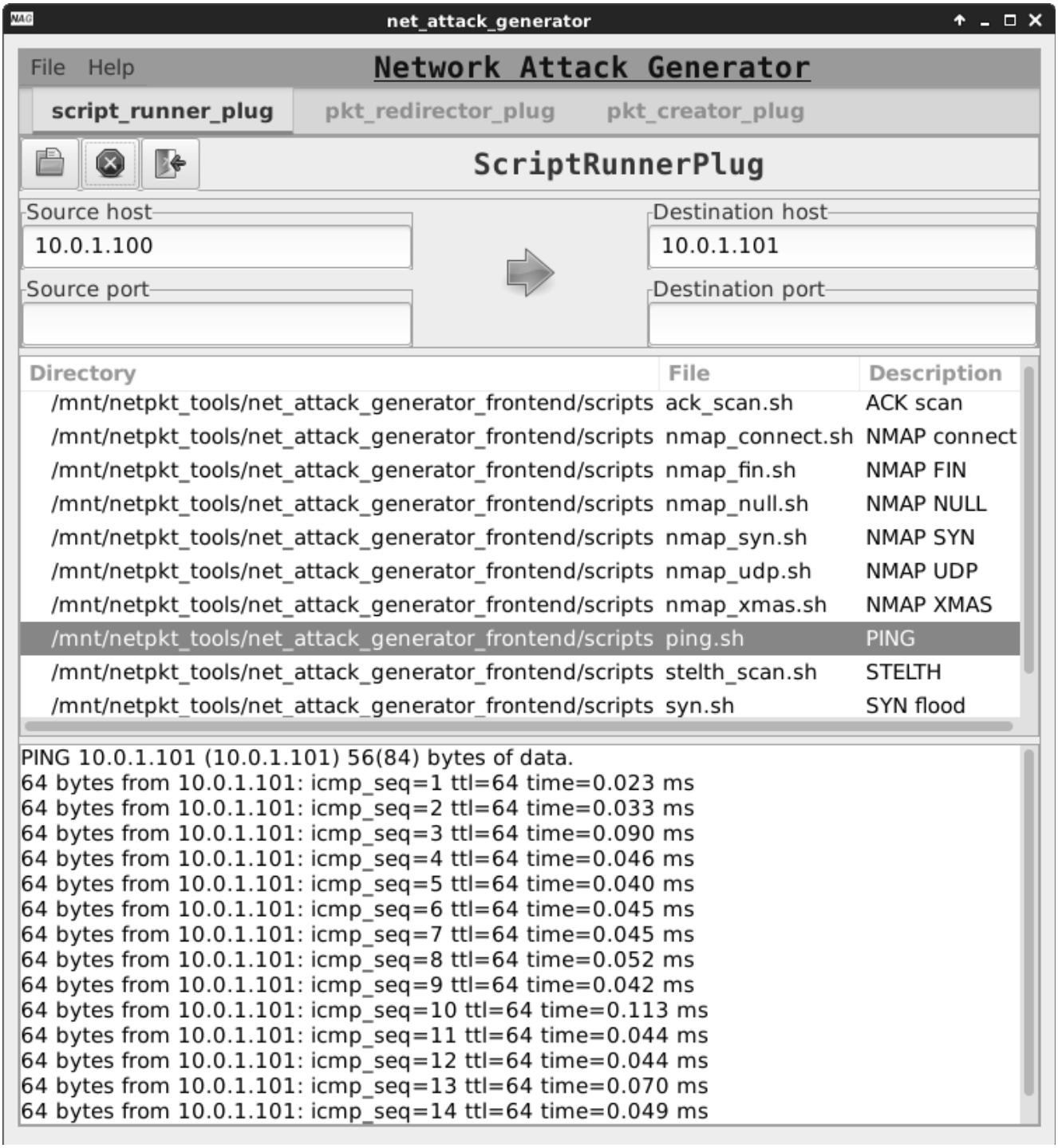


Рисунок 3.9 — Генератор сетевых атак (вкладка запуска скриптов)

2. PktRedirectorPlug (рисунок 3.10). Данная вкладка предназначена для воспроизведения сетевого трафика из pcap-файла или перенаправления трафика из одного сетевого интерфейса в другой. Среди основных опций присутствует возможность сохранения сетевых дампов, BPF-фильтрации и изменения IP-адресов в захваченных пакетах с автоматическим пересчетом контрольных сумм на IP-, TCP-, UDP- и ICMP-уровнях;

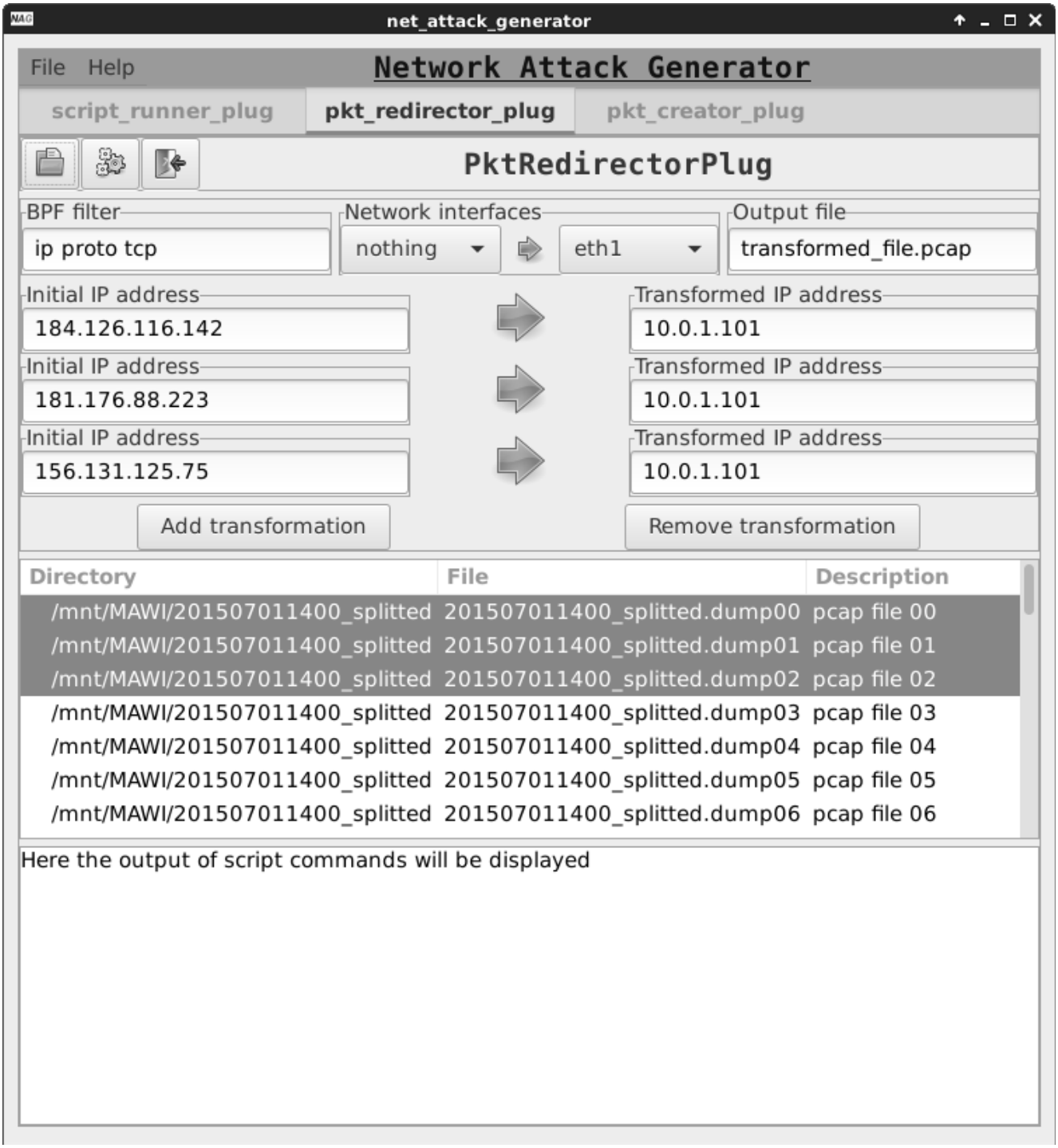


Рисунок 3.10 — Генератор сетевых атак (вкладка чтения сетевых дампов)

3. PktCreatorPlug (рисунок 3.11). Данная вкладка предназначена для создания IP-пакетов версии 4. Возможна установка любых полей пакета, начиная с канального уровня и заканчивая уровнем пользовательских данных. Среди дополнительных опций можно назвать возможность отправки пакета в заданный сетевой интерфейс, указания числа дублей, автоматического вычисления контрольных сумм в пакете.

Рисунок 3.11 — Генератор сетевых атак (вкладка создания сетевых пакетов)

Каждая из представленных заглушек — это бинарный исполняемый ELF-файл, который может быть запущен из панели меню, расположенной в верхней части frontend-интерфейса. Заглушка встраивается в основное окно приложения при помощи вызова add_id с аргументом идентификатора заглушки над элементом гнезда (или через вызов родительского конструктора заглушки с переданным в него аргументом идентификатора гнезда). При разработке frontend-интерфейса использовались АЯВУ C++, библиотеки boost (libboost_program_options.so, libboost_regex.so, libboost_filesystem.so, libboost_system.so, libboost_thread.so) и графическая библиотека Gtk-3. Объем исходного кода генератора сетевых атак — 2770 непустых строк без учета комментариев. За счет наличия большого числа системнозависимых вызовов в нижележащих компонентах и библиотеках разработанное программное средство предназначено только для запуска в ОС Linux. Исходный код представлен в следующих файлах:

- net_attack_gen.h и net_attack_gen.cpp. Файлы с описанием и реализацией класса NetAttackGenWindow (основного окна приложения), производного от Gtk::Window;
- base_plug.h и base_plug.cpp. Файлы с описанием и реализацией абстрактного базового класса BasePlug (контейнера Gtk-заглушек), производного от Gtk::Plug;
- script_runner_plug.h и script_runner_plug.cpp. Файлы с описанием и реализацией класса ScriptRunnerPlug, производного от BasePlug;
- pkt_redirector_plug.h и pkt_redirector_plug.cpp. Файлы с описанием и реализацией класса PktRedirectorPlug, производного от BasePlug;
- pkt_creator_plug.h и pkt_creator_plug.cpp. Файлы с описанием и реализацией класса PktCreatorPlug, производного от BasePlug;
- interprocess_communicator.h и interprocess_communicator.cpp. Файлы с описанием и реализацией класса InterProcessCommunicator с поддержкой именованных каналов, файловых сокетов и разделяемой памяти;
- log.h. Файл с директивами для вывода отладочной информации;
- Makefile. Файл с правилами для сборки приложения.

3.4 Результаты экспериментов

Представленные в данном разделе эксперименты по обнаружению сетевых атак при помощи адаптивных подходов, а именно методов ВИ, направлены на практическое исследование трех теоретических положений: алгоритма генетико-конкурентного обучения сети Кохонена, эволюционной модели искусственной иммунной системы и методики иерархической гибридизации интеллектуальных детекторов, а также архитектуры СОА. Используемый для выполнения экспериментов программный макет представлен в разделе А.9 приложения А. При проведении первых четырех экспериментов использовался созданный автором настоящего диссертационного исследования набор данных, который включал два класса аномальных соединений и фоновый белый трафик. Для генерации атак применялось описанное в разделе 3.3 программное средство вместе с многочисленными скриптами (рисунок 3.9), в частности с инструментами nmap и hping3, запущенными в различных режимах и предназначенными соответственно для выполнения атак типа сканирования хостов и отказа в обслуживании. Основой для создания векторов признаков, описывающих нормальный трафик, послужил набор данных MAWILab Data Set 2015/07/01, который был предобработан компонентом pkt_reader, запущенным внутри frontend-интерфейса (рисунок 3.10).

Эксперимент 1. На рисунках Г.1, Г.2, Г.3, Г.4, Г.5, Г.6 изображены графики ошибки векторного квантования для двух типов алгоритма конкурентного обучения (генетического и стандартного) в зависимости от номера эпохи обучения и радиуса окрестности нейрона-победителя. Обучение производилось с использованием различных настроек и на разных подвыборках, соответствующих определенному классу соединения: легитимному трафику (normal), сканированию хостов (scan) и отказу в обслуживании (synflood). Каждая точка этих графиков получена в результате усреднения ошибок векторного квантования на 100 прогонах, причем начальная и каждый раз случайная инициализация весовых коэффициентов сетей Кохонена для обоих алгоритмов была идентичной.

Во всех трех случаях для данного эксперимента использовалась стратегия генетической оптимизации на основе механизма рулетки без операторов мутации и инверсии. Для демонстрации работоспособности алгоритма генетико-конкурентного обучения и его превосходства над стандартной реализацией этого алгоритма были использованы различные характеристики сети Кохонена, представленные в таблице 8 и соответствующие вышепредставленным рисункам. В частности, за счет использования метода главных компонент размерность входного вектора варьировалась в пределах от 10 до 20 с шагом 5. Форма выходной решетки — квадрат, число нейронных узлов, размещенных внутри него, — от 100 до 400. Генетическая оптимизация для модифицированного алгоритма применялась после каждой эпохи обучения сети Кохонена.

Таблица 8 — Используемые в экспериментах характеристики сети Кохонена

Рисунки	Класс сетевого соединения	Начальный параметр скорости обучения κ_0	Мощность обучающей выборки	Размерность входного вектора	Размер выходной решетки	Число эпох обучения	Начальный радиус окружения
Г.1, Г.2	normal	0.7	900	20	15×15	400	21.21
Г.3, Г.4	scan	0.3	1000	10	20×20	300	28.28
Г.5, Г.6	synflood	0.01	400	15	10×10	200	14.14

На рисунках Г.1, Г.3 наблюдается убывание ошибки $\frac{1}{M} \cdot E(\boldsymbol{w_{11}}, \ldots, \boldsymbol{w_{IJ}})$, которое чередуется со скачкообразным локальным ростом, особенно ярко проявляющимся на начальных эпохах обучения. Как уже говорилось в разделе 2.3, такая ситуация может быть обусловлена тем, что вначале осуществляется грубая настройка весов сети Кохонена, а затем выполняется их тонкая адаптация по мере убывания мультипликативного коэффициента скорости обучения $\kappa(t)$ с течением времени t. Чтобы избавиться от подобного эффекта волнообразного спада и придать кривой ошибки векторного квантования требуемый гладкий спуск,

можно использовать малое значение параметра κ_0, к примеру $\kappa_0 = 0.01$. Такое поведение кривой, обладающей на большей части временного участка плавным мотонным убыванием, представлено на рисунке Г.5.

Из рисунка Г.3 видно, что добавление разработанной стохастической оптимизации весов нейронов сети Кохонена в алгоритм конкурентного обучения привело к сокращению числа эпох в этом алгоритме. Наименьшее значение ошибки векторного квантования, достигаемое в случае стандартного алгоритма после выполнения заключительной 299-ой эпохи, составляет 0.00127. Для оптимизированного генетического алгоритма это же значение ошибки векторного квантования выполняется уже после 189-ой эпохи. Тем самым введение стратегий генетической оптимизации позволило сократить функционирование алгоритма на 110 эпох обучения.

Для графика, представленного на рисунке Г.1, количество эпох было уменьшено на 74. Величина ошибки векторного квантования, численно равная 0.01721, достигается на 398-ой эпохе обучения в случае стандартного алгоритма и на 324-ой эпохе обучения в случае модифицированного алгоритма.

Эксперимент 2. Следующий набор экспериментов направлен на исследование эффективности функционирования иммунной системы. Состав сгенерированного тестового набора данных — 30000 экземпляров (10000 примеров каждого класса). Эксперименты выполнялись для 16 контрольных подмножеств этого набора данных (мощность тестового набора данных изменялась от 7500 до 30000 элементов с шагом 1500). Каждое из этих подмножеств использовалось 100 раз в качестве контрольного для обученных на различных подвыборках иммунных систем. На рисунках Г.7, Г.8, Г.9, Г.10, Г.11, Г.12, Г.13, Г.14 схематически изображены показатели эффективности иммунной системы в зависимости от размера подмножества сгенерированного тестового набора данных.

Данные эксперименты выполнялись для иммунной системы, состоящей из шести детекторов (по три детектора на каждый аномальный класс соединения), причем размерность выходной решетки соответствующих сетей Кохонена выбиралась произвольным образом в пределах четырех целочисленных двумерных интервалов (10–15 $\times$ 10–15, 15–20 $\times$ 15–20, 20–25 $\times$ 20–25, 25–30 $\times$ 25–30). Каждая из четырех конфигураций иммунной системы обучалась 100 раз с использованием различных подвыборок (мощностью 4500 примеров), равнообъемных для каждого класса соединений (1500 экземпляров каждого класса соединений) и случайным образом извлеченных из обучающего набора данных (объемом 6750 векторов признаков соединений). Каждый столбик гистограммы получен в результате усреднения 100 значений соответствующего показателя эффективности (TPR, FPR, CCR, ICR, GPR, GCR); вертикальной чертой, пронизывающей каждый столбик, обозначены пределы изменения этой величины. Размерность входного вектора — 25 компонент, количество эпох обучения — $T = 500$, начальный параметр скорости обучения — $\kappa_0 = 0.1$.

Вычисление параметров эффективности осуществлялось с использованием двух значений порогов: h_{ij}^- и h_{ij}. Во втором случае корректировка порогов h_{ij}^-, вычисленных для нейронов (i,j) на шаге 9 алгоритма генетико-конкурентного обучения, в сети Кохонена, настроенной для распознавания l-ого класса аномальных соединений, осуществлялась по следующему правилу:

$$
\begin{aligned}
h_{ij} = &\begin{cases} h_{ij}^- - \beta \cdot \left(h_{ij}^- - h_{ij}^+\right), \text{ если } \forall \boldsymbol{x_k} \in C_0 \bigcap \mathcal{X}_{\mathcal{C}}^{(LS)}\ d_{ijk}^{-1} < h_{ij}^- \\ h_{ij}^-, \text{ если } \exists \boldsymbol{x_k} \in C_0 \bigcap \mathcal{X}_{\mathcal{C}}^{(LS)}\ d_{ijk}^{-1} \geqslant h_{ij}^- \text{ (ложное срабатывание)}, \end{cases} \\
h_{ij}^- = &\min_{\boldsymbol{x_k} \in C_l \bigcap \mathcal{X}_{\mathcal{C}}^{(LS)}} \left\{ d_{ijk}^{-1} \,|(i,j) = F_{IJ}(\boldsymbol{x_k}) \right\}, \\
h_{ij}^+ = &\max_{\boldsymbol{x_k} \in C_0 \bigcap \mathcal{X}_{\mathcal{C}}^{(LS)}} \left\{ d_{ijk}^{-1} | (i,j) = F_{IJ}(\boldsymbol{x_k}) \right\}, \\
\beta = &\frac{1}{2}.
\end{aligned}
$$

В результате проведенных экспериментов было установлено, что в случае использования процедуры корректировки порогов наибольшее значение показа-

теля TPR варьировалось в пределах от 83.09% до 97.14% (рисунок Г.7) при уменьшении размера тестового набора данных с 30000 элементов до 7500 элементов с шагом -1500. В то же время значение этого показателя без корректировки значения порога не превышало отметки 77.64% (рисунок Г.7). Тем самым за счет внедрения вышепредставленной процедуры корректировки порогов было достигнуто увеличение показателя TPR на 19.5% на контрольном множестве из 7500 элементов и на 47.59% на контрольном множестве из 30000 элементов, при этом максимальное значение показателя ложных срабатываний FPR осталось прежним (2.4%) для первого контрольного множества и поднялось всего лишь на 0.11% для второго контрольного множества. Наибольшее значение показателя CCR с использованием процедуры корректировки порогов на контрольном множестве из 7500 и 30000 элементов составило соответственно почти 97.51% и 88.18%, что на 13.14% и 31.36% превышает аналогичный показатель, вычисленный без использования процедуры корректировки порогов на соответствующих множествах данных. Одновременно с этим максимальное значение показателя ICR опустилось на 6% и 8.02%.

Вычисленные с использованием порога h_{ij} и представленные на рисунке Г.8 параметры обобщающей способности при обнаружении (GPR) и при классификации (GCR) достигали максимальной отметки 100% на наборе, состоящем из 750 уникальных элементов, с сохранением средних значений показателей ложных срабатываний и некорректной классификации на уровне 0.05% и 17.21%. Для этой же конфигурации иммунной системы на наборе данных, состоящем из 23214 уникальных элементов, максимальные значения показателей GPR и GCR составляли 80.91% и 86.84%, а средние значения показателей FPR и ICR равнялись 0.01% и 25.59%.

Эксперимент 3. Более детальная корректировка порогов позволяет повысить эффективность иммунной системы с сохранением приемлемого уровня компромиссного баланса между показателями TPR (GPR), CCR (GCR) и FPR, ICR. Графики, отражающие зависимость этих показателей от параметра β показаны на рисунках Г.15, Г.16, Г.17, Г.18.

На каждом из указанных выше рисунков изображены шесть кривых, которые соответствуют минимальным, максимальным и средним значениям рассматриваемых показателей, вычисленных для 100 предварительно обученных иммунных систем (10–15×10–15) с шагом 0.01 для параметра β. Ранее значение параметра β, влияющего на порог активации иммунных детекторов, устанавливалось фиксированным, а именно 0.5, теперь этот параметр рассматривается как подлежащий настройке: его оптимальное значение вычисляется в соответствии со следующими условиями: $TPR - FPR + CCR - ICR \to \max\limits_{\beta=0,0.01,\ldots,1}$ и $GPR - FPR + GCR - ICR \to \max\limits_{\beta=0,0.01,\ldots,1}$ и равняется 0.6. В то время как для параметра $\beta = 0.5$ средние значения показателей эффективности на контрольном множестве из 30000 элементов имеют следующие значения: $TPR = 67.322\%$, $FPR = 0.266\%$, $CCR = 77.695\%$, $ICR = 22.305\%$, для параметра $\beta = 0.6$ аналогичные параметры принимают следующие значения: $TPR = 77.282\%$, $FPR = 0.285\%$, $CCR = 84.244\%$, $ICR = 15.757\%$. Тем самым за счет использования подобной процедуры дополнительной корректировки порога активации показатели TPR и CCR на данном тестовом множестве выросли на 9.96% и 6.549% соответственно, а показатель ложных срабатываний увеличился на несущественные 0.019%. В то же время максимальные значения этих показателей, соответствующих параметру $\beta = 0.6$, составляют величины: $TPR = 98.15\%$, $FPR = 0.78\%$, $CCR = 98.153\%$, что на 15.06%, 0.07% и 9.973% превышает аналогичные показатели, соответствующие параметру $\beta = 0.5$. Применяя аналогичные расчеты к контрольному множеству, состоящему из 23214 уникальных элементов, не встречашихся в процессе обучения иммунных детекторов, приходим к выводу, что вычисленные для него средние значения показателей GPR, FPR и GCR увеличились на 12.881%, 0.025% и 8.481%, а максимальные значения этих же показателей выросли на 17.622%, 0.09% и 11.734%.

В описанных выше экспериментах параметр β использовался как глобальный для всей иммунной системы; безусловно, показатели ее эффективности могут быть увеличены путем введения персональных параметров β для задания порога активации каждого отдельного иммунного детектора.

Стоит отметить, что кривые, изображенные на рисунках Г.16 и Г.18, являются симметричными относительно горизонтальной прямой 50%. Это объясняется отсутствием конфликтных случаев классификации ($CCR + ICR = 100\%$), которое достигается за счет использования нечетного числа детекторов для распознавания каждого класса аномальных соединений и разработанного подхода разрешения таких ситуаций.

Эксперимент 4. Обучение и тестирование $4 \times 100 = 400$ иммунных систем осуществлялось с использованием двух режимов: однопоточного и многопоточного. Первый режим запускался каждый десятый раз для каждой конфигурации иммунной системы, во втором режиме создавалось шесть потоков, в каждом из которых исполнялся иммунный детектор. Временные затраты этих процессов представлены в таблице 9.

Таблица 9 — Временные затраты обучения и тестирования иммунных систем

Режим		Конфигурация иммунной системы			
		10–15 × 10–15	15–20 × 15–20	20–25 × 20–25	25–30 × 25–30
Обучение: количество эпох — 500 (сек.)					
одно-	min	928.975	2347.179	4695.537	8484.724
поточный	max	1307.356	3267.926	6009.498	10 738.129
режим	avg	1062.930	2700.014	5536.395	9854.751
много-	min	181.128	429.896	971.027	1699.392
поточный	max	341.705	903.677	1738.795	3145.876
режим	avg	257.685	613.349	1264.316	2254.569
Тестирование: вход — один 25-мерный вектор признаков (мкс.)					
одно-	min	53	59	67	84
поточный	max	2733	2572	2343	2124
режим	avg	128.108	141.521	151.801	151.698
много-	min	137	139	139	135
поточный	max	11 358	24 494	15 849	16 203
режим	avg	321.932	312.347	311.109	313.334

В среднем при использовании многопоточного режима во время обучения иммунных систем было достигнуто ускорение настройки ее детекторов, в 4.319 раза превышающее однопоточный режим функционирования. С другой стороны, необходимо учитывать, что распараллеливание является эффективным способом оптимизации только той задачи, для которой время создания потока несопоставимо мало по сравнению с временем выполнения одной из выделенных в ней подзадач, запущенной внутри этого потока. Из таблицы 9 видно, что многопоточный режим исполнения задачи анализа одного входного вектора уступает однопоточному режиму в терминах времяемкости.

Эксперимент 5. Для оценки эффективности гибридных подходов использовался открытый набор данных DARPA 1998. Этот набор данных широко используется для исследования методов обнаружения сетевых атак, однако он подвержен острой негативной критике, вызванной его „синтетической“ природой [154]. В частности, в [150] отмечается, что классификацию между аномальным и нормальным трафиком возможно выполнять, используя всего лишь несколько полей из IP-заголовка (к примеру, TTL). В разработанной автором настоящего диссертационного исследования методике подобные поля не используются, т.к. их значения никоим образом не отражают факта наличия или отсутствия сетевой аномалии (к примеру, пакеты в рамках одной TCP-сессии в реальной сети Internet могут пойти разными маршрутами). Параметры, рассматриваемые в данном диссертационном исследовании для классификации сетевых соединений, являются по большей части именно статистическими, а набор данных DARPA 1998 используется для проведения экспериментов ввиду его удобства: вместе с pcap-файлами поставляются CSV-файлы с метками соединений. Кроме того, устаревшие типы атак были исключены (например, переполнение буфера в почтовой серверной службе или атака teardrop, реализующая старую уязвимость в стеке TCP/IP во время выполнения дефрагментации пакетов), а рассмотрению подлежат именно сетевые аномалии, характеризующиеся некоторыми отклонениями в значениях статистических параметров. Тем самым было выделено 7 классов сетевых соединений: 6 аномальных классов: DoS (отказ в обслужива-

нии, а именно neptune, smurf); Probe (сканирование хостов и портов, а именно ipsweep, nmap, portsweep, satan); и 1 нормальный класс: normal.

Детальный состав обучающего и тестового множеств, сгенерированных на основе набора данных DARPA 1998, представлен в таблице 10. Уникальные записи трех классов аномальных сетевых соединений, а именно neptune, smurf и portsweep, полностью встречаются в процессе обучения, тем самым оба показателя обобщения (GPR и GCR) вычисляются в данном эксперименте на разреженном множестве меток, соответствующих четырем оставшимся классам. Отметим, что эксперимент 7 не имеет этого недостатка и использует равно пропорциональное для каждого класса разбиение обучающего и тестового множеств.

Таблица 10 — Обучающее и тестовое множества на основе DARPA 1998

Множество		Класс соединения						
		neptune	smurf	ipsweep	nmap	portsweep	satan	normal
обучающее множество	общее количество записей	1000	1000	1000	1000	1000	1000	1000
		7000						
	количество уникальных записей	789	524	1000	1000	649	1000	1000
		5962						
тестовое множество	общее количество записей	2445	1572	4302	3405	1947	26616	60826
		101113						
	количество уникальных записей	789	524	1264	1015	649	8870	40622
		53733						
	количество уникальных записей без обучающих	0	0	264	15	0	7870	39622
		47771						

При формировании этого набора данных были использованы бинарные сетевые трейсы Training Data DARPA 1998, собранные в среду первой недели, понедельник второй недели, вторник второй недели, понедельник третьей недели, среду третьей недели, пятницу третьей недели, вторник четвертой недели и среду четвертой недели. В таблицах 16, 17, 18, 19 представлены вычисленные на этом наборе данных показатели эффективности адаптивных классификаторов.

Обучение коллектива классификаторов выполнялось десять раз для каждой из четырех низкоуровневых схем комбинирования детекторов (one-vs-all, one-vs-one, CBT, DAG), причем для чистоты проведения экспериментов обучение агрегирующих композиционных правил, таких как метод взвешенного голосования (МВГ) и метод многоярусной укладки (ММУ), выполнялось для идентично обученного коллектива базовых классификаторов. Стоит отметить, что метод мажоритарного голосования (ММГ) и метод Фикса—Ходжеса (МФХ)[5] не требуют подобных процедур настройки их внутренних параметров. Из таблицы 16 видно, что МФХ демонстрирует наилучшие результаты как в терминах корректности обнаружения и классификации, так и в терминах ложных срабатываний, однако наибольшее значение показателей $TPR = 99.866\%$ и $CCR = 99.557\%$ принадлежит ММУ. Анализируя данные из этой же таблицы, приходим к выводу, что в сравнении с базовым классификатором, представленным ННС и имеющим наилучшее усредненное значение показателя $TPR = 99.543\%$ среди остальных базовых классификаторов, МФХ превосходит его на 0.253% по корректности обнаружения и на 0.336% по уровню ложных срабатываний. В сравнении с МНС, демонстрирующей наилучшее усредненное значение показателя $CCR = 98.86\%$, МФХ превосходит этот классификатор на 0.551% по корректности классификации и на 0.408% по уровню некорректной классификации. В данных экспериментах число групп, объединяющих базовые решатели, равнялось 3.

Эксперимент 6. Для исследования устойчивости агрегирующих композиций в состав коллектива классификационных правил были введены 2 типа классификаторов: случайный классификатор (СК) и ложный классификатор (ЛК). В то время как выходом первого классификатора является случайный набор меток классов, среди которых может оказаться и правильный, второй классификатор заведомо исключает эту возможность: его показатель корректности классификации всегда равен 0. Таблицы 20, 21, 22, 23 содержат показатели эффективности коллектива классификаторов, в котором РБФ и РНС заменены на СК и ЛК.

[5]Одним из ограничений МФХ является необходимость сохранения обучающего набора данных, элементы и классы которого будут использоваться в процессе анализа входного вектора для поиска ближайших к нему k соседей (в экспериментах данный параметр k был выбран равным 5). По мнению автора, этот процесс не является обучением.

В экспериментах, соответствующих таблицам 20, 21, 22, 23, повторного обучения не производилось, что подтверждается сохранением прежних значений показателей эффективности, вычисленных для базовых классификаторов и представленных в таблицах 16, 17, 18, 19.

На примере таблицы 20 рассмотрим устойчивость композиций, объединяющих пять базовых классификаторов, среди которых два являются ошибочными, в сравнении с полученными ранее результатами. Показатель TPR, вычисленный для ММГ в среднем снизился на 0.926%, а для показателя FPR ММГ сместился на 0.087% в большую сторону, сохранив при этом по-прежнему самый низкий уровень ложных срабатываний. Показатель CCR, вычисленный для МВГ, в среднем снизился на 0.011%, на это же значение увеличился показатель ICR; а значения показателей TPR и FPR увеличились на 0.112% и 0.228%. МФХ, несмотря на добавление ошибочных классификаторов в коллектив решателей, сохранил за собой лидирующее положение как по показателю TPR, так и по показателю CCR. Для него значение первого показателя не изменилось, а значение второго показателя снизилось в среднем на 0.145%. Кроме того, для МФХ значения показателей FPR и ICR увеличились на 0.135% и 0.073% соответственно. Как и следовало ожидать, наиболее чувствительным к введению новых классификаторов в состав коллектива правил оказался ММУ (имевший ранее самый внушительный показатель $CCR = 99.557\%$), для которого среднее значение показателя CCR снизилось с 99.07% до 44.739%.

Подобная ситуация, связанная с введением ошибочных классификаторов, может наблюдаться, когда злоумышленник изменил настройки СОА таким образом, что вместо „легитимных“ плагинов были загружены ошибочные. Кроме того, это может быть вызвано случаями, когда один из базовых классификаторов (например, иммунная система), динамически переобучаясь, потерял способность к обнаружению прежних сетевых аномалий.

Эксперимент 7. Поскольку главная цель, поставленная в настоящем диссертационном исследовании, — это уменьшение значения функционала эмпирического риска, т.е. повышение показателей GCR (CCR), то были проведе-

ны дополнительные эксперименты для более детального и точного вычисления несмещенной оценки этой характеристики. С этой целью использовалась пятиблочная кросс-валидация [181]. Набор данных $\bar{\mathcal{X}}_{\mathcal{C}}^{(TS)}$, содержащий $\bar{M}^*$=53733 неповторяющиеся записи сетевых соединений, был разбит на пять дизъюнктных подмножеств $\overset{(1)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)},\ldots,\overset{(5)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)}$, у которых $\#\overset{(1)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \approx \cdots \approx \#\overset{(5)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)}$. Причем в каждом множестве $\overset{(k)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)}$ встречаются элементы всех 7 классов соединений таким образом, что подвыборка, соответствующая каждому определенному l-ому классу, имеет примерно равный размер внутри каждого множества $\overset{(k)}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} : \#\overset{(1)}{\bar{\mathcal{X}}}_{\{C_l\}}^{(TS)} \approx \cdots \approx \overset{(5)}{\bar{\mathcal{X}}}_{\{C_l\}}^{(TS)}$ ($k = 1,\ldots,5$, $l = 0,\ldots,6$). Обучающая и контрольная выборки были взяты в отношении $3 : 2$. Процесс обучения базовых классификаторов выполнялся $3 \times C_5^3 = 30$ раз (для каждой из четырех низкоуровневых схем комбинирования детекторов) при помощи множеств $\left\{ \overset{(a)_p}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \cup \overset{(b)_p}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \cup \overset{(c)_p}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \right\}_{p=1}^{3}$, где $a,b,c \in \mathbb{N} \wedge 1 \leqslant a < b < c \leqslant 5$. В зависимости от этих множеств контрольное множество составляется следующим образом: $\left\{ \overset{(d)_p}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \cup \overset{(e)_p}{\bar{\mathcal{X}}}_{\mathcal{C}}^{(TS)} \right\}_{p=1}^{3}$, где $d,e \in \mathbb{N} \setminus \{a,b,c\} \wedge 1 \leqslant d < e \leqslant 5$, и на каждом из этих множеств вычисляются значения показателей i-ого базового классификатора $GPR_{i(de)_p}^{(\text{БК})}$, $FPR_{i(de)_p}^{(\text{БК})}$, $GCR_{i(de)_p}^{(\text{БК})}$, $ICR_{i(de)_p}^{(\text{БК})}$ ($i = 1,\ldots,5$) и j-ого коллективного правила $GPR_{j(de)_p}^{(\text{КП})}$, $FPR_{j(de)_p}^{(\text{КП})}$, $GCR_{j(de)_p}^{(\text{КП})}$, $ICR_{j(de)_p}^{(\text{КП})}$ ($j = 1,\ldots,4$). Подобное разбиение множества $\bar{\mathcal{X}}_{\mathcal{C}}^{(TS)}$ выполнялось трижды ($p = 1,2,3$), причем каждый раз обеспечивалась случайная генерация содержимого его десяти подмножеств. Минимальные, максимальные и средние значения показателя GPR, соответствующие i-ому базовому классификатору и j-ому коллективному правилу, определяются следующим образом: $\min\limits_{p=1,2,3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{i(de)_p}^{(\text{БК})}$, $\max\limits_{p=1,2,3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{i(de)_p}^{(\text{БК})}$, $\frac{1}{3} \cdot \sum\limits_{p=1}^{3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{i(de)_p}^{(\text{БК})}$ и $\min\limits_{p=1,2,3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{j(de)_p}^{(\text{КП})}$, $\max\limits_{p=1,2,3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{j(de)_p}^{(\text{КП})}$, $\frac{1}{3} \cdot \sum\limits_{p=1}^{3} \frac{1}{10} \cdot \sum\limits_{1 \leqslant d < e \leqslant 5} GPR_{j(de)_p}^{(\text{КП})}$. Аналогичным образом вычисляются величины остальных показателей. Полученные значения разностей каждого из этих показателей схематически изображены на рисунках Г.19, Г.20, Г.21, Г.22, Г.23, Г.24, Г.25, Г.26.

Наибольшее среднее значение разности показателей GPR и FPR принадлежит МФХ и составляет величину $GPR - FPR = 99.726\%$ (рисунок Г.21), что превышает на 0.044% значение аналогичного показателя, вычисленного для ННС ($GPR - FPR = 99.682\%$) и являющегося максимальным среди демонстрируемых базовыми классификаторами. Наилучшее среднее значение разности показателей GCR и ICR представлено на рисунке Г.20 ($GCR - ICR = 99.579\%$): МФХ превосходит МНС в терминах этого показателя на 0.438%. Для рисунка Г.19 этот разрыв между МФХ и лучшим представителем среди базовых классификаторов становится более существенным и равняется 0.142% и 1.275% соответственно для показателя $GPR - FPR$, вычисленного для МФХ и МОВ, и показателя $GCR - ICR$, вычисленного для МФХ и МНС. Подобный небольшой выигрыш обусловлен тем, что используемые в экспериментах базовые классификаторы имеют и без того высокие показатели эффективности, а также они сами по себе представляют коллектив нескольких решателей. В частности, в схеме one-vs-one каждый базовый классификатор представляет собой коллектив 3 групп, каждая из которых содержит $C_7^2 = 21$ детектор (рисунок 2.17), что в общей сложности составляет 63 детектора внутри каждого базового классификатора. С учетом того, что в приведенных выше экспериментах каждое верхнеуровневое коллективное правило объединяет вместе 5 базовых классификаторов, общее количество детекторов может составлять величину 315. Все это, как может показаться, должно приводить к существенному потреблению системных и временных ресурсов, однако использование в плагинах низкоуровневого машинного кода, реализованного на C, позволяет устранить эту проблему.

Эксперимент 8. В таблице 24 представлены индикаторы ресурсопотребления (время обучения классификаторов и их тестрирования, загрузка процессора и размер потребляемой памяти[6]), замеры которых производились при помощи инструментов top и pmap (с частотой 0.1 сек.) во время обучения и тестирования адаптивных классификаторов.

[6]ОС Linux обладает сложным механизмом управления памятью [109, 114, 191], и точный размер потребляемой процессом памяти может быть вычислен только с помощью его полной изоляции от других процессов (к примеру, при помощи запуска внутри valgrind), поэтому здесь под реальным размером понимается размер разделяемой и закрытой памяти, выделенной процессу.

Во время обучения потребление оперативной памяти для однопоточного режима не превышало 134.219 MB, а для многопоточного режима — 506.09 MB. Максимальное время обучения составило около 3 часов 24 минут (для схемы one-vs-all) в однопоточном режиме и около 1 часа 57 минут в многопоточном режиме (для схемы one-vs-all). Для многопоточного режима во время тестирования в каждом потоке анализировался только один экземпляр сетевого соединения при помощи одного из бинарных классификаторов, что привело к увеличению времени обработки этого вектора по сравнению с однопоточным режимом.

Эксперимент 9. Зависимость усредненной по сорока обучающим множествам меры информативности ς от числа выбранных главных компонент $\widehat{n}$ представлена на рисунке 3.12.

На рисунке 3.13 показан результат отображения обучающей выборки, содержащей семь классов, в двумерное пространство при помощи метода главных компонент.

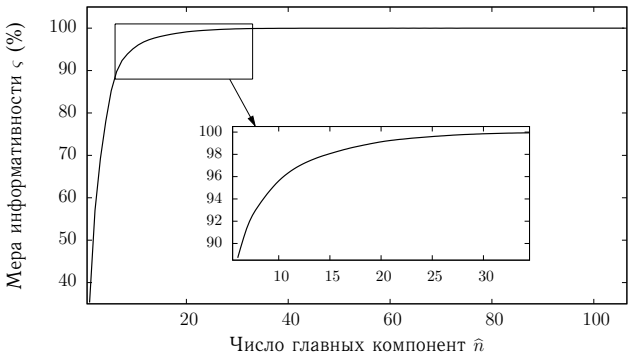


Рисунок 3.12 — Зависимость меры информативности от числа главных компонент на наборе данных DARPA 1998

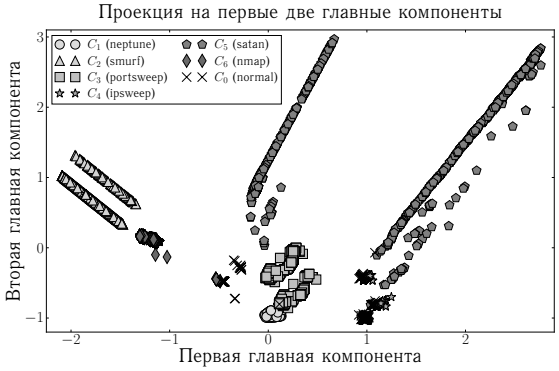


Рисунок 3.13 — Отображение обучающей выборки в двумерное пространство при помощи метода главных компонент

Из рисунка 3.12 видно, что после получения приблизительно 30 первых главных компонент уже не наблюдается существенного прироста меры информативности, и график кривой практически полностью вырождается в постоянную функцию. В экспериментах значение порогового параметра ε выбиралось равным 99.9%, что соответствовало 33 первым главным компонентам.

Все ранее представленные эксперименты выполнялись на вычислительной машине, имеющей следующую конфигурацию: гостевая ОС Debian 8.8 (8 cores, 16085 MB RAM), запущенная в виртуальном окружении KVM внутри хостовой ОС Debian 8.5 (2 physical processors Intel(R) Xeon(R) CPU E5-2630, 6 cores per processor, 12 threads per processor, 129168 MB RAM, 1200–3100 MHz CPU frequency, 32KB L1 cache, 256 KB L2 cache, 15360 KB L3 cache, 3.7 TB HDD).

Эксперимент 10. Дополнительно была произведена оценка эффективности разработанного сенсора netpcap_sensor в сравнении с такими СОА, как Snort, Suricata и Bro (версии этих СОА представлены в разделе А.1 приложения А). С этой целью каждая СОА запускалась 100 раз в режиме offline-чтения сетевых дампов. Во время очередного запуска СОА (с выдержанной трехминутной паузой после предыдущего запуска) анализировался pcap-файл 200608241400.dump (из набора данных MAWILab Data Set), обрезанный при помощи компонента pkt_reader ровно до 10 млн. пакетных записей (размер результирующего файла равен 743489222 байта, средний размер пакета равен 58.348923 байт). Анализатор netpcap_sensor компилировался с включенными директивами IP-дефрагментации, TCP-реассемблирования, хранения 5-секундной истории захваченных сетевых пакетов, хеширования сетевых соединений вместе с заполнением указателей на соединения, имеющие одинаковый IP-адрес источника (назначения), и запускался без опции вычисления сетевых параметров. Размер списка ip_conn_pool, содержащего предварительно выделенную память для размещения новых соединений, — $MAX\_IP\_POOL\_SIZE = 300000$, размер таблицы ip_conn_table — $MAX\_IP\_TABLE\_SIZE = 100000$. Сравнение осуществлялось по следующим характеристикам: (1) $T^{(real)}$ — общее время функционирования; (2) $T^{(proc)}$ — время обработки пакетов; (3) $V^{(pkts)}$ — количество обрабатываемых килопакетов за единицу времени; (4) $V^{(data)}$ — количество обрабатываемых килобайтов за единицу времени; (5) $L^{(CPU)}$ — загрузка CPU; (6) $C^{(virt)}$ — размер потребляемой виртуальной памяти; (7) $C^{(resd)}$ — размер потребляемой резидентной памяти; (8) $C^{(real)}$ — размер потребляемой реальной памяти. В таблице 25 приведены измеренные значения этих характеристик для каждой СОА.

Конфигурация используемой в этом эксперименте машины приведена в разделе A.10 приложения А. Здесь, как и в эксперименте 8, использовались инструменты top и pmap, которые выполняли мониторинг параметров процесса СОА через каждые 0.1 секунды.

По всем из представленных параметров сенсор netpcap_sensor превосходит другие рассмотренные программные продукты. В частности, средняя скорость обработки пакетов сенсором составляет 324.756 kpkts/sec., что более чем в 1.5 раза превышает аналогичный показатель, вычисленный для Suricata в режиме single, при этом размер потребляемой сенсором реальной памяти почти в 1.5 раза ниже, чем у указанной СОА. Наличие нулевых значений в строке $L^{(CPU)}$ соответствует тем моментам времени, когда процесс запущенной СОА (Snort, Suricata) оказывался в состоянии непрерываемого (бесперебойного) сна (uninterruptible sleep). В этом состоянии процесс ожидает I/O сигнал от драйвера дискового или сетевого устройства. Анализ полученных данных показал, что подобные значения характерны во время старта программы, когда выполняются системные вызовы инициализации дескрипторов устройств и файлов. Нулевые значения, которые представлены в строках, соответствующих размеру потребляемой памяти различных типов, достигаются только непосредственно перед завершением функционирования СОА (Suricata и Bro). Из таблицы 25 видно, что благодаря разработанному механизму преаллокации памяти сенсор netpcap_sensor сразу же с первой секунды старта выделяет практически весь необходимый объем памяти, который потребуется для дальнейшего хранения хешированных записей, содержащих данные о сетевых соединениях.

Результаты экспериментов доказывают, что разработанная СОА удовлетворяет заявленным в разделе 1.4 функциональным и нефункциональным требованиям, а также требованиям [57, 59, 95, 96, 103], предъявляемым к вычислительно интеллектуальным системам, а именно она (1) является адаптивной в терминах ВИ (эксперимент 1), (2) обладает способностью обобщения (эксперименты 2, 3, 5, 7), (3) устойчива к наличию шумов (эксперимент 6), (4) характеризуется высокой скоростью функционирования (эксперименты 4, 8, 9, 10).

3.5 Предложения по применению разработанного модельно-методического аппарата для построения СОА

Представленные в данном разделе предложения по применению разработанного модельно-методического аппарата для построения СОА носят в основном технический характер и заключаются в следующем:

1. Распараллеливание алгоритмов обнаружения сетевых атак.

Данный прием заключается в разбиении исходной задачи на несколько непересекающихся подзадач, каждая из которых будет исполняться независимо на отдельном процессоре или ядре. В настоящем диссертационном исследовании выполнен экспериментальный анализ различных параллельных алгоритмов, встроенных в ядро СОА, а именно алгоритмов сигнатурного поиска подстрок, алгоритма обучения иммунной системы и алгоритма каскадного обучения дерева классификаторов.

2. Использование машинного кода в качестве основы для построения СОА.

Разработка СОА — это системная задача, которая требует извлечения наибольшей производительности из ресурсов оборудования. Это может достигаться только за счет использования компиляторов, учитывающих аппаратные характеристики архитектуры ОС и способных воспроизводить низкоуровневый машинный код. Использование скриптовых языков и компиляторов, генерирующих управляемый код, может вызывать существенные временные задержки, неприемлемые в рамках данной задачи. Такие АЯВУ, как Python и Java, не позволяют достичь необходимой скорости в обработке данных. Кроме того, Python-скрипты не подходят для создания многопоточного кода и не подлежат распараллеливанию внутри POSIX-потоков: за счет механизма GIL (Global Interpreter Lock) возникают ошибки, связанные с выделением памяти во время вызова функции PyObject_Malloc. Для устранения этой проблемы применяются средства для обхода GIL, к примеру, компилятор Cython помогает реализовывать низкоуровневые модули с возможностью их загрузки в среду исполнения Python [21].

3. Оптимизация доступа к памяти при разработке компонентов СОА.

В настоящем диссертационном исследовании был изложен прием предварительной аллокации памяти, реализованный в сетевом анализаторе и пакетном балансировщике. Данный прием поволяет снизить временные затраты во время непосредственного анализа сетевых пакетов за счет хранения свободных записей IP-потоков в связном однонаправленном списке ip_conn_pool.

4. Смешивание обучающих данных.

Необходимо формировать обучающие множества таким образом, чтобы они содержали, по возможности, равнообъемные выборки для каждого класса соединений. С этой целью можно дублировать некоторые элементы из маломощных классов для получения „сбалансированного“ обучающего множества. Кроме того, рекомендуется использовать прием смешивания обучающих данных, т.е. чередовать элементы одного класса с элементами другого класса, причем иногда дополнительный выигрыш удается получить при перестановке данных внутри обучающего множества от одной эпохи обучения к другой. Все это позволит избежать тех случаев, когда модель переобучается или обучается лучше для распознавания наиболее представительных классов, но при этом забывает элементы маломощных классов.

5. Автоматизация процесса выполнения экспериментов.

При проведении экспериментов автором настоящего диссертационного исследования использовались Bash- и Perl-скрипты как для запуска компонентов СОА, так и для обработки результатов их функционирования. Непрерывный процесс генерации набора данных и выполнения экспериментов занял более двух месяцев, в течение которых было сгенерировано более 27 ГБ лог-файлов.

6. Разбиение множества классов атак для распознавания детекторами.

При разработке СОА использовался подход „снизу-вверх“: вначале сужалось множество исследуемых классов аномальных соединений, затем для каждого из них строился специфичный классификатор, учитывающий особенности этого сетевого соединения. Такой подход, в отличие от других, позволяет разделять исследуемое множество атак между различными классификаторами.

Другие предложения, направленные на усовершенствование функционирования СОА, перечислены в разделе А.8 приложения А.

Дальнейшие технические улучшения и дополнения разработанной СОА могут включать следующие пункты:

1. Разработка модифицированного модуля предотвращения сетевых атак.

На данный момент блокировка подозрительных сетевых соединений выполняется при помощи правил стандартного (встроенного в ядро Linux) межсетевого экрана — iptables. Запросы на настройку заложенных в него цепочек правил выполняются на сенсорах и поступают от коллектора в случае выявления потенциально аномальных сетевых потоков. Улучшение данного модуля видится в добавлении средств активного противодействия нелегальному сетевому трафику, поступающему от злоумышленника.

2. Добавление алгоритмов обучения адаптивных классификаторов с распараллеливанием на уровне видеокарты.

В настоящем диссертационном исследовании были разработаны библиотеки, содержащие функции параллельного поиска шаблонных подстрок с использованием технологии CUDA. Направление дальнейших улучшений может быть связано с применением этой же технологии к алгоритмам обучения адаптивных классификаторов. Введение такой модификации позволит, в частности, сократить время адаптации классификаторов к параметрам измененного сетевого окружения.

3. Добавление других методов ВИ в ядро классификации объектов.

Алгоритмы роевого интеллекта — одно из звеньев ВИ, которое, как и генетические алгоритмы, широко используется для решения задач оптимизации. Дальнейшее направление может быть связано с разработкой таких алгоритмов для их включения в разработанную систему классификации объектов.

Выводы по главе 3

В главе 3 решены следующие задачи:

1. Разработана архитектура сетевой распределенной СОА. Архитектура СОА является двухуровневой: первый уровень обеспечивает первичный анализ отдельных пакетов и сетевых соединений при помощи сигнатурного анализа, на втором уровне осуществляется обработка агрегированных сетевых потоков данных при помощи адаптивных классификаторов.
2. Разработан стенд генерации сетевых атак. Данный стенд состоит из двух компонент: асинхронного прозрачного прокси-сервера TCP-сессий и frontend-интерфейса генератора сетевых атак.
3. Выполнены эксперименты по оценке предложенных модели, алгоритма и методики, проведено экспериментальное сравнение разработанного сетевого сенсора с характеристиками открытых СОА. Результаты экспериментов подтвердили то, что заявленные в разделе 1.4 функциональные и нефункциональные требования, а также требования, предъявляемые к вычислительно интеллектуальным системам, выполняются для разработанной СОА.

Заключение

В диссертационной работе решена задача разработки модельно-методического аппарата для обнаружения аномальных сетевых соединений на основе гибридизации методов ВИ. **Основные результаты** сводятся к следующему:

1. Разработана модель искусственной иммунной системы на базе эволюционного подхода. Модель характеризуется наличием двухуровневой процедуры обучения и тестирования и позволяет учитывать динамическую природу сетевого трафика посредством переобучения иммунных детекторов как с течением времени, так и в ответ на выявленные в сетевом трафике аномалии. Встроенный в модель алгоритм группового обнаружения сетевых атак позволяет снизить число конфликтных случаев их классификации. Результаты проведенных экспериментальных исследований подтвердили отсутствие таких ситуаций.

2. Разработан алгоритм генетико-конкурентного обучения сети Кохонена. Отличительной особенностью алгоритма является наличие процесса имитации эволюции «мертвых» нейронов, который достигается за счет использования нескольких стратегий генетической оптимизации весовых коэффициентов сети Кохонена. Дополненная оптимизация позволяет сократить количество эпох обучения при достижении заданной максимальной величины ошибки векторного квантования. В частности, для случая класса scan число эпох обучения было уменьшено на 110 по сравнению со стандартной реализацией алгоритма.

3. Разработана методика иерархической гибридизации бинарных классификаторов. Методика предоставляет возможность строить многоуровневые схемы с произвольной вложенностью классификаторов друг в друга и их «ленивым» подключением в процессе анализа входного вектора. В случае применения пятиблочной кросс-валидации и низкоуровневой схемы one-vs-all для комбинирования детекторов экспериментально было получено увеличение показателя $GCR-ICR$ на 1.275%. Программная реализация этой методики в виде интеллектуального яд-

ра классификации объектов может быть использована обособленно от СОА при решении общих задач классификации объектов.

4. Разработана архитектура распределенной СОА, построенной на основе гибридизациии методов ВИ и сигнатурного анализа. СОА характеризуется возможностью горячей вставки исполняемого кода за счет загрузки плагинов, представленных в виде бинарных библиотек и встраиваемых динамически в ядро СОА. Ее событийно-ориентированный анализатор предоставляет интерфейс для доступа как к полям отдельных пакетов, так и к параметрам реассемблированных сетевых соединений. Эксперименты показали, что скорость обработки сетевых потоков при помощи разработанной СОА более чем в 1.5 раза превышает аналогичный показатель, демонстрируемый другими решениями.

Представленный модельно-методический аппарат позволяет повысить эффективность функционирования СОА и может быть использован как основа для построения систем классификации объектов.

Рекомендации по применению разработанного модельно-методического аппарата для построения СОА включают применение механизмов распараллеливания для обнаружения сетевых атак, использование машинного кода в качестве основы для СОА, разработку подходов, направленных на оптимизацию доступа к памяти. Полученные в диссертационной работе экспериментальные результаты подтверждают целесообразность использования этих рекомендаций.

Перспективы дальнейшей разработки темы заключаются в расширении списка вычисляемых сетевых параметров для учета особенностей, свойственных новым типам аномалий, и совершенствовании предложенной архитектуры СОА с целью ее адаптации под современные классы сетевых атак.

Полученные результаты соответствуют п. 13 паспорта специальностей ВАК (технические науки) «Принципы и решения (технические, математические, организационные и др.) по созданию новых и совершенствованию существующих средств защиты информации и обеспечения информационной безопасности» по специальности «05.13.19 Методы и системы защиты информации, информационная безопасность».

Список сокращений и условных обозначений

АЯВУ	алгоритмический язык высокого уровня
БД	база данных
ВИ	вычислительный ителлект
ИИ	искусственный интеллект
ОС	операционная система
ПО	программное обеспечение
СОА	система обнаружения атак
СУБД	система управления базами данных
ФС	файловая система
ACK	ACKnowledgement flag (флаг квитирования в заголовке TCP-пакета)
API	Application Programming Interface (интерфейс прикладного программирования)
ARP	Address Resolution Protocol (протокол отображения адресов)
BPF	Berkeley Packet Filter (пакетный фильтр BSD Unix)
CPU	Central Processing Unit (центральное процессорное устройство)
CSV	Comma-Separated Values (значения, разделенные запятыми)
DHCP	Dynamic Host Configuration Protocol (протокол динамического конфигурирования хостов)
DNS	Domain Name System (система доменных имен)
DoS	Denial of Service (отказ в обслуживании)
FDDI	Fiber Distributed Data Interface (волоконно-оптический распределенный интерфейс данных)
FIN	FINish flag (флаг окончания передачи данных в TCP-соединении)
FTP	File Transfer Protocol (протокол передачи файлов)
GPU	Graphics Processing Unit (графическое процессорное устройство)
HTTP	HyperText Transfer Protocol (протокол передачи гипертекста)
ICMP	Internet Control Message Protocol (протокол межсетевых управляющих сообщений)
IMAP	Internet Message Access Protocol (протокол для доступа к сообщениям электронной почты)

IP	Internet Protocol (межсетевой протокол)
ISN	Initial Sequence Number (начальный позиционный номер сегментов в TCP-соединении)
JSON	JavaScript Object Notation (объектная нотация JavaScript)
LALR	Look-Ahead Left-to-Right parsing (восходящий алгоритм синтаксического анализа)
OSI	Open Systems Interconnection (взаимодействие открытых систем)
POP3	Post Office Protocol Version 3 (протокол почтового отделения, версия 3)
PPP	Point-to-Point Protocol (протокол обмена по непосредственным связям „точка-точка“)
PSH	PuSH flag (TCP-флаг „проталкивания“ данных в пользовательский буфер приложения)
RAM	Random Access Memory (память с произвольным доступом)
RARP	Reverse Address Resolution Protocol (протокол обратного отображения адресов)
RPC	Remote Procedure Call (вызов удаленных процедур)
RST	ReSeT flag (флаг принудительного разрыва TCP-соединения)
SIP	Session Initiation Protocol (протокол установления сеанса)
SLIP	Serial Line Internet Protocol (межсетевой протокол для последовательных линий)
SMTP	Simple Mail Transfer Protocol (простой протокол передачи почты)
SSH	Secure Shell (безопасная оболочка)
SSL	Secure Sockets Layer (уровень защищенных сокетов)
SYN	SYNchronize sequence number flag (флаг синхронизации позиционных номеров TCP-соединения)
TCP	Transmission Control Protocol (протокол управления передачей)
Telnet	remote terminal protocol (протокол удаленного терминала)
TLS	Transport Layer Security (безопасность транспортного уровня)
TTL	Time-To-Live (время жизни IP-пакета)
UDP	User Datagram Protocol (протокол дейтаграмм пользователя)
URG	URGent pointer flag (флаг срочности в заголовке TCP-пакета)
XML	eXtensible Markup Language (расширяемый язык разметки)

Список литературы

1. *Айвазян, С. А.* Теория вероятностей и прикладная статистика, 2-е издание / С. А. Айвазян, В. С. Мхитарян. Т. 1. — М. : Юнити-Дана, 2001. — 656 с.
2. *Айвазян, С. А.* Прикладная статистика: классификация и снижение размерности / С. А. Айвазян, В. М. Бухштабер, И. С. Енюков, Л. Д. Мешалкин. — М. : Финансы и статистика, 1989. — 607 с.
3. *Альбертс, Б.* Молекулярная биология клетки: в трех томах / Б. Альбертс, Д. Брей, Д. Льюис, М. Рэфф, К. Робертс, Д. Уотсон. — М.–Ижевск : НИЦ «Регулярная и хаотическая динамика», Институт компьютерных исследований, 2013. — 2764 с.
4. *Борисов, В. В.* Нечеткие модели и сети / В. В. Борисов, В. В. Круглов, А. С. Федулов. — М. : Горячая линия–Телеком, 2007. — 284 с.
5. *Браницкий, А. А.* Анализ и классификация методов обнаружения сетевых атак / А. А. Браницкий, И. В. Котенко // Труды СПИИРАН. — 2016. — Т. 2, № 45. — С. 207–244.
6. *Вапник, В. Н.* Теория распознавания образов. Статистические проблемы обучения / В. Н. Вапник, А. Я. Червоненкис. — М. : Наука, 1974. — 416 с.
7. *Вапник, В. Н.* Восстановление зависимостей по эмпирическим данным / В. Н. Вапник. — М. : Наука, 1979. — 448 с.
8. *Головко, В. А.* Нейронные сети: обучение, организация и применение / В. А. Головко. — М. : Издательское предприятие редакции журнала „Радиотехника“, 2001. — 256 с.
9. *Гонсалес, Р.* Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. — М. : Техносфера, 2005. — 1072 с.
10. *Городецкий, В. И.* Методы и алгоритмы коллективного распознавания: обзор / В. И. Городецкий, С. В. Серебряков // Труды СПИИРАН. — 2006. — Т. 1, № 3. — С. 139–171.

11. *Журавлев, Ю. И.* Об алгебраическом подходе к решению задач распознавания или классификации / Ю. И. Журавлев // Проблемы кибернетики. — 1978. — Т. 33. — С. 5–68.
12. *Каллан, Р.* Основные концепции нейронных сетей / Р. Каллан. — М. : Издательский дом „Вильямс“, 2001. — 287 с.
13. *Колмогоров, А. Н.* О представлении непрерывных функций нескольких переменных суперпозициями непрерывных функций меньшего числа переменных / А. Н. Колмогоров // Докл. АН СССР. Т. 114. — 1957. — С. 953–956.
14. *Кольман, Я.* Наглядная биохимия / Я. Кольман, К.-Г. Рем. — М. : Мир, 2004. — 469 с.
15. *Кондратьев, А. А.* Методологическое обеспечение интеллектуальных систем защиты от сетевых атак / А. А. Кондратьев, А. А. Талалаев, И. П. Тищенко, В. П. Фраленко, В. М. Хачумов // Современные проблемы науки и образования. — 2014. — № 2.
16. *Кохонен, Т.* Самоорганизующиеся карты, 2-е издание / Т. Кохонен. — М., 2014. — 655 с.
17. *Круглов, В. В.* Искусственные нейронные сети. Теория и практика / В. В. Круглов, В. В. Борисов. — М. : Горячая линия–Телеком, 2001. — 382 с.
18. *Леоненков, А. В.* Нечеткое моделирование в среде MATLAB и fuzzyTECH / А. В. Леоненков. — СПб. : БХВ-Петербург, 2005. — 736 с.
19. *Лукацкий, А. В.* Обнаружение атак / А. В. Лукацкий. — СПб. : БХВ-Петербург, 2003. — 608 с.
20. *Люгер, Д. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание / Д. Ф. Люгер. — М. : Издательский дом „Вильямс“, 2003. — 864 с.
21. *Маккинни, У.* Python и анализ данных / У. Маккинни. — М. : ДМК Пресс, 2015. — 482 с.

22. *Мартыненко, Б. К.* Языки и трансляции / Б. К. Мартыненко. — СПб. : Изд-во СПбГУ, 2004. — 231 с.
23. *Норткат, С.* Обнаружение нарушений безопасности в сетях, 3-е издание / С. Норткат, Д. Новак. — М. : Издательский дом „Вильямс“, 2003. — 448 с.
24. *Осовский, С.* Нейронные сети для обработки информации / С. Осовский. — М. : Финансы и статистика, 2002. — 344 с.
25. *Паклин, Н. Б.* Бизнес-аналитика: от данных к знаниям, 2-е издание / Н. Б. Паклин, В. И. Орешков. — СПб. : Питер, 2013. — 704 с.
26. *Писаревский, Б. М.* О математике, математиках и не только, 2-е издание / Б. М. Писаревский, В. Т. Харин. — М. : БИНОМ. Лаборатория знаний, 2012. — 302 с.
27. *Пол, У.* Иммунология / У. Пол, А. Сильверстайн, М. Купер. Т. 1. — М. : Мир, 1987. — 476 с.
28. *Половко, И. Ю.* Анализ функциональных требований к системам обнаружения вторжений / И. Ю. Половко, О. Ю. Пескова // Известия Южного федерального университета. Технические науки. — 2014. — 2 (151). — С. 86–92.
29. *Рассел, С.* Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. — М. : Издательский дом „Вильямс“, 2017. — 1408 с.
30. *Растригин, Л. А.* Метод коллективного распознавания. Библиотека по автоматике. Вып. 615 / Л. А. Растригин, Р. Х. Эренштейн. — М. : Энергоиздат, 1981. — 80 с.
31. *Ройт, А.* Иммунология / А. Ройт, Д. Бростофф, Д. Мейл. — М. : Мир, 2000. — 592 с.
32. *Рутковская, Д.* Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. — М. : Горячая линия–Телеком, 2006. — 452 с.
33. *Стивенс, У. Р.* UNIX: взаимодействие процессов / У. Р. Стивенс. — СПб., 2002. — 576 с.

34. *Стивенс, У. Р.* UNIX: разработка сетевых приложений / У. Р. Стивенс. — СПб. : Питер, 2003. — 1088 с.
35. *Стивенс, У. Р.* Протоколы TCP/IP. Практическое руководство / У. Р. Стивенс. — СПб. : Невский Диалект, 2003. — 672 с.
36. *Стренг, Г.* Линейная алгебра и ее применения / Г. Стренг. — М. : Мир, 1980. — 454 с.
37. *Тейлор, Д.* Биология / Д. Тейлор, Н. Грин, У. Стаут. Т. 2. — М. : Мир, 2004. — 436 с.
38. *Тэрано, Т.* Прикладные нечеткие системы / Т. Тэрано, К. Асаи, М. Сугэно. — М. : Мир, 1993. — 368 с.
39. *Хайкин, С.* Нейронные сети: полный курс, 2-е издание / С. Хайкин. — М. : Издательский дом „Вильямс“, 2006. — 1104 с.
40. *Шаньгин, В. Ф.* Информационная безопасность компьютерных систем и сетей / В. Ф. Шаньгин. — М. : ИД «Форум»: ИНФРА-М, 2008. — 416 с.
41. *Ярилин, А. А.* Иммунология / А. А. Ярилин. — М. : ГЭОТАР-Медиа, 2010. — 752 с.
42. *Ясницкий, Л. Н.* Введение в искусственный интеллект / Л. Н. Ясницкий. — М. : Издательский дом „Академия“, 2005. — 176 с.
43. *Abraham, A.* Distributed intrusion detection systems: a computational intelligence approach / A. Abraham, J. Thomas // Applications of Information Systems to Homeland Security and Defense. — IGI Global, 2006. — Pp. 107–137.
44. *Agarwal, B.* Hybrid approach for detection of anomaly network traffic using data mining techniques / B. Agarwal, N. Mittal // Procedia Technology. — 2012. — Vol. 6. — Pp. 996–1003.
45. *Aho, A. V.* Algorithms for finding patterns in strings / A. V. Aho // Handbook of Theoretical Computer Science. Algorithms and Complexity. — 1990. — Pp. 255–300.

46. *Aho, A. V.* Efficient string matching: an aid to bibliographic search / A. V. Aho, M. J. Corasick // Communications of the ACM. — 1975. — Vol. 18, no. 6. — Pp. 333–340.
47. *Aickelin, U.* The danger theory and its application to artificial immune systems / U. Aickelin, S. Cayzer //. — 2002. — Pp. 141–148.
48. *Aickelin, U.* Sensing danger: Innate immunology for intrusion detection / U. Aickelin, J. Greensmith // Information Security Technical Report. — 2007. — Vol. 12, no. 4. — Pp. 218–227.
49. *Albin, E.* A comparative analysis of the snort and suricata intrusion-detection systems / E. Albin: MA thesis / Albin, Eugene. — Naval Postgraduate School, Monterey CA, 2011. — 49 pp.
50. *Amini, M.* Effective intrusion detection with a neural network ensemble using fuzzy clustering and stacking combination method / M. Amini, J. Rezaeenour, E. Hadavandi // Journal of Computing and Security. — 2014. — Vol. 1, no. 4. — Pp. 293–305.
51. *Anderson, J. P.* [et al.] Computer security threat monitoring and surveillance / J. P. Anderson: tech. rep. — Feb. 1980.
52. *Arboleda, A. F.* SnortTM diagrams for developers / A. F. Arboleda, C. E. Bedón [Электронный ресурс]. — 2005. — URL: http://artemisa.unicauca.edu.co/~cbedon/snort/snortdevdiagrams.pdf (visited on 08/01/2017).
53. *Barbara, D.* Detecting novel network intrusions using bayes estimators / D. Barbara, N. Wu, S. Jajodia // In Proceedings of the 2001 SIAM International Conference on Data Mining. — SIAM. 2001. — Pp. 1–17.
54. *Barford, P.* Characteristics of Network Traffic Flow Anomalies / P. Barford, D. Plonka // In Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement. — ACM. 2001. — Pp. 69–73.
55. *Barford, P.* A signal analysis of network traffic anomalies / P. Barford, J. Kline, D. Plonka, A. Ron // In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement. — ACM. 2002. — Pp. 71–82.

56. *Beale, J.* Snort 2.1 intrusion detection / J. Beale, B. Caswell. — Syngress, 2004. — 751 pp.
57. *Bezdek, J. C.* What is computational intelligence? / J. C. Bezdek // Computational Intelligence: Imitating Life. — 1994. — Pp. 1–12.
58. *Bezdek, J. C.* (Computational) Intelligence: What's in a Name? / J. C. Bezdek // IEEE Systems, Man, and Cybernetics Magazine. — 2016. — Vol. 2, no. 2. — Pp. 4–14.
59. *Bezdek, J. C.* On the relationship between neural networks, pattern recognition and intelligence / J. C. Bezdek // International journal of approximate reasoning. — 1992. — Vol. 6, no. 2. — Pp. 85–107.
60. *Bezdek, J. C.* The history, philosophy and development of computational intelligence (How a simple tune became a monster hit) / J. C. Bezdek // Computational intelligence. Oxford, UK: Eolss Publishers. — 2013. — Pp. 1–21.
61. *Bishop, C. M.* Pattern recognition and machine learning / C. M. Bishop. — Springer, 2006. — 738 pp.
62. *Boyce, C. A. P.* A Comparison of Four Intrusion Detection Systems for Secure E-Business / C. A. P. Boyce, A. N. Zincir-Heywood // In Proceedings of the International Conference on Electronic Commerce Research. — 2003. — Pp. 302–314.
63. *Boyer, R. S.* A fast string searching algorithm / R. S. Boyer, J. S. Moore // Communications of the ACM. — 1977. — Vol. 20, no. 10. — Pp. 762–772.
64. *Braden, R.* RFC 1071. Computing the Internet Checksum / R. Braden, D. Borman, C. Partridge [Электронный ресурс]. — 1988. — URL: https://www.ietf.org/rfc/rfc1071.txt (visited on 08/01/2017).
65. *Branitskiy, A.* Network attack detection based on combination of neural, immune and neuro-fuzzy classifiers / A. Branitskiy, I. Kotenko // In Proceedings of the 18th IEEE International Conference on Computational Science and Engineering (IEEE CSE2015). — IEEE. Oct. 2015. — Pp. 152–159.

66. *Bretscher, P.* A theory of self-nonself discrimination / P. Bretscher, M. Cohn // Science. — 1970. — Vol. 169, no. 3950. — Pp. 1042–1049.
67. *Brindasri, S.* Evaluation Of Network Intrusion Detection Using Markov Chain / S. Brindasri, K. Saravanan // International Journal on Cybernetics & Informatics (IJCI). — 2014. — Vol. 3, no. 2. — Pp. 11–20.
68. *Brockwell, P. J.* Introduction to time series and forecasting / P. J. Brockwell, R. A. Davis. — Springer, 2002. — 434 pp.
69. *Brownlee, J.* Clever algorithms: nature-inspired programming recipes / J. Brownlee. — 2011. — 441 pp.
70. *Bunke, H.* Hybrid methods in pattern recognition / H. Bunke, A. Kandel. Vol. 47. — World Scientific, Series in Machine Perception and Artificial Intelligence, 2002. — 496 pp.
71. *Burnet, S. F. M.* [et al.] The clonal selection theory of acquired immunity / S. F. M. Burnet. — Vanderbilt University Press Nashville, 1959. — 209 pp.
72. *Cannady, J.* Artificial neural networks for misuse detection / J. Cannady // National information systems security conference. — 1998. — Pp. 368–381.
73. *Celada, F.* A computer model of cellular interactions in the immune system / F. Celada, P. E. Seiden // Immunology today. — 1992. — Vol. 13, no. 2. — Pp. 56–62.
74. *Cercone, N.* Ten years of computational intelligence / N. Cercone, G. McCalla // Computational Intelligence. — 1994. — Vol. 10, no. 4. — Pp. 1–6.
75. *Chambers, L. D.* Practical handbook of genetic algorithms: complex coding systems / L. D. Chambers. Vol. 3. — CRC press, 1998. — 572 pp.
76. *Chandrasekhar, A. M.* Intrusion detection technique by using k-means, fuzzy neural network and SVM classifiers / A. M. Chandrasekhar, K. Raghuveer // In Proceedings of International Conference on Computer Communication and Informatics (ICCCI). — IEEE. 2013. — Pp. 1–7.
77. *Chan-Tin, E.* The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems / E. Chan-Tin, D. Feldman, N. Hopper,

Y. Kim // In proceeding of 5th International ICST Conference on Security and Privacy in Communication Networks. — Springer. 2009. — Pp. 448–458.

78. *Chan-Tin, E.* The frog-boiling attack: Limitations of secure network coordinate systems / E. Chan-Tin, V. Heorhiadi, N. Hopper, Y. Kim // ACM Transactions on Information and System Security (TISSEC). — 2011. — Vol. 14, no. 3.
79. *Cheng, Y.* RFC 7413. TCP fast open / Y. Cheng, J. Chu, S. Radhakrishnan, A. Jain [Электронный ресурс]. — 2014. — URL: https://www.ietf.org/rfc/rfc7413.txt (visited on 08/01/2017).
80. *Cid, D. B.* Log analysis using OSSEC / D. B. Cid. — 2007. — URL: http://ossec.net/ossec-docs/auscert-2007-dcid.pdf (visited on 08/01/2017).
81. *Craenen, B. C.* Computational intelligence / B. C. Craenen, A. E. Eiben // Encyclopedia of Life Support Sciences. — 2002. — Pp. 1–14.
82. *Cybenko, G.* Approximation by superpositions of a sigmoidal function / G. Cybenko // Mathematics of Control, Signals, and Systems (MCSS). — 1989. — Vol. 2, no. 4. — Pp. 303–314.
83. *Dasgupta, D.* Immunity-based intrusion detection system: a general framework / D. Dasgupta // In Proceeding of the 22nd National Information Systems Security Conference (NISSC). Vol. 1. — 1999. — Pp. 147–160.
84. *Dasgupta, D.* Immunological computation: theory and applications / D. Dasgupta, F. Nino. — CRC press, 2008. — 277 pp.
85. *Day, D.* A performance analysis of snort and suricata network intrusion detection and prevention engines / D. Day, B. Burns // In Proceedings of the Fifth International Conference on Digital Society (ICDC), Gosier, Guadeloupe. — 2011. — Pp. 187–192.
86. *De Castro L. N.* The clonal selection algorithm with engineering applications / L. N. De Castro, F. J. Von Zuben // In Proceedings of GECCO (Workshop on Artificial Immune Systems and Their Applications). — July 2000. — Pp. 36–39.

87. *De Castro L. N.* Artificial immune systems as a novel soft computing paradigm / L. N. De Castro, J. I. Timmis // Soft computing. — 2003. — Vol. 7, no. 8. — Pp. 526–544.
88. *De Castro L. N.* Artificial immune systems: a new computational intelligence approach / L. N. De Castro, J. Timmis. — Springer, 2002. — 357 pp.
89. *De Castro L. N.* Artificial immune systems: Part I–basic theory and applications / L. N. De Castro, F. J. Von Zuben: tech. rep. — Dec. 1999. — 95 pp.
90. *Debar, H.* RFC 4765. The intrusion detection message exchange format (IDMEF) / H. Debar, D. A. Curry, B. S. Feinstein [Электронный ресурс]. — 2007. — URL: https://www.ietf.org/rfc/rfc4765.txt (visited on 08/01/2017).
91. *Debar, H.* Towards a taxonomy of intrusion-detection systems / H. Debar, M. Dacier, A. Wespi // Computer Networks. — 1999. — Vol. 31, no. 8. — Pp. 805–822.
92. *Denning, D. E.* An intrusion-detection model / D. E. Denning // IEEE Transactions on software engineering. — 1987. — Feb. — No. 2. — Pp. 222–232.
93. *Drucker, H.* Support vector regression machines / H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, V. Vapnik // In Proceedings of the 1996 Conference on Advances in neural information processing systems. — MIT Press. 1997. — Pp. 155–161.
94. *Duch, W.* What is computational intelligence and what could it become / W. Duch // Computational Intelligence, Methods and Applications Lecture Notes NTU, Singapour. — 2003.
95. *Eberhart, R. C.* Computational intelligence: concepts to implementations / R. C. Eberhart, Y. Shi. — Morgan Kaufmann, 2007. — 496 pp.
96. *Eberhart, R. C.* Computational intelligence PC tools / R. C. Eberhart, P. K. Simpson, R. Dobbins, R. W. Dobbins. — Academic Press Professional, 1996. — 464 pp.

97. *Engelbrecht, A. P.* Computational intelligence: an introduction / A. P. Engelbrecht. — John Wiley & Sons, 2007. — 597 pp.
98. *Ennert, M.* Testing of IDS model using several intrusion detection tools / M. Ennert, E. Chovancová, Z. Dudláková // Journal of Applied Mathematics and Computational Mechanics. — 2015. — Vol. 14, no. 1. — Pp. 55–62.
99. *Fahlman, S. E.* Faster learning variations on back-propagation: An empirical study / S. E. Fahlman // In Proceedings of the Connectionist Models Summer School. — Morgan Kaufmann. 1988. — Pp. 38–51.
100. *Fausett, L.* Fundamentals of neural networks: architectures, algorithms, and applications / L. Fausett. — Prentice-Hall, 1994. — 461 pp.
101. *Fix, E.* Discriminatory analysis-nonparametric discrimination: consistency properties / E. Fix, J. L. Hodges Jr: tech. rep. / California Univ Berkeley. — 1951. — 21 pp.
102. *Fogel, D. B.* Evolutionary computation: toward a new philosophy of machine intelligence / D. B. Fogel. Vol. 1. — John Wiley & Sons, 2006. — 296 pp.
103. *Fogel, D. B.* Review of Computational Intelligence: Imitating Life [Book Reviews] / D. B. Fogel // Proceedings of the IEEE. — 1995. — Vol. 83, no. 11. — Pp. 1588–1592.
104. *Forrest, S.* Self-nonself discrimination in a computer / S. Forrest, A. S. Perelson, L. Allen, R. Cherukuri // In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA. IEEE Computer. — IEEE. Society Press, 1994. — Pp. 202–212.
105. *Funahashi, K.-I.* On the approximate realization of continuous mappings by neural networks / K.-I. Funahashi // Neural networks. — 1989. — Vol. 2, no. 3. — Pp. 183–192.
106. *Galar, M.* An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes / M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera // Pattern Recognition. — 2011. — Vol. 44, no. 8. — Pp. 1761–1776.

107. *Ghorbani, A. A.* Network intrusion detection and prevention: concepts and techniques / A. A. Ghorbani, W. Lu, M. Tavallaee. Vol. 47. — Springer Science & Business Media, 2009. — 212 pp.

108. *Golovko, V.* Principles of neural network artificial immune system design to detect attacks on computers / V. Golovko, M. Komar, A. Sachenko // In Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science. — IEEE. 2010. — P. 237.

109. *Gorman, M.* Understanding the Linux virtual memory manager / M. Gorman. — Prentice Hall Upper Saddle River, 2004. — 768 pp.

110. *Govindarajan, M.* Intrusion detection using an ensemble of classification methods / M. Govindarajan, R. M. Chandrasekaran // In Proceeding of the World Congress on Engineering and Computer Science. Vol. 1. — Oct. 2012. — Pp. 459–464.

111. *Greensmith, J.* Dendritic cells for SYN scan detection / J. Greensmith, U. Aickelin // In Proceedings of the 9th annual conference on genetic and evolutionary computation. — ACM. 2007. — Pp. 49–56.

112. *Greensmith, J.* Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection / J. Greensmith, U. Aickelin, S. Cayzer // In Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS-05). Vol. 3627. — 2005. — Pp. 153–167.

113. *Greensmith, J.* Information fusion for anomaly detection with the dendritic cell algorithm / J. Greensmith, U. Aickelin, G. Tedesco // Information Fusion. — 2010. — Vol. 11, no. 1. — Pp. 21–34.

114. *Gregg, B.* Systems performance: enterprise and the cloud / B. Gregg. — Prentice Hall, 2013. — 792 pp.

115. *Haugeland, J.* Artificial intelligence: The very idea / J. Haugeland. — MIT press, 1989. — 299 pp.

116. *Hay, A.* OSSEC host-based intrusion detection guide / A. Hay, D. Cid, R. Bray. — Syngress, 2008. — 416 pp.

117. *Hayes, P.* Turing test considered harmful / P. Hayes, K. Ford // In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. — 1995. — Pp. 972–977.

118. *He, H.-T.* Detecting anomalous network traffic with combined fuzzy-based approaches / H.-T. He, X.-N. Luo, B.-L. Liu // Advances in Intelligent Computing. — 2005. — Pp. 433–442.

119. *Hensel, F.* Flow-based and Packet level-based Intrusion Detection as Complementary Concepts / F. Hensel: Diploma Thesis / Hensel, Fabian. — University of Zurich, Department of Informatics, 2008. — 160 pp.

120. *Hofmeyr, S. A.* An interpretative introduction to the immune system / S. A. Hofmeyr // Design principles for the immune system and other distributed autonomous systems. — 2001. — Vol. 3. — Pp. 3–26.

121. *Hofmeyr, S. A.* Architecture for an artificial immune system / S. A. Hofmeyr, S. Forrest // Journal of Evolutionary computation. — 2000. — Vol. 8, no. 4. — Pp. 443–473.

122. *Hofmeyr, S. A.* An immunological model of distributed detection and its application to computer security / S. A. Hofmeyr: PhD thesis / Hofmeyr, Steven Andrew. — PhD thesis, University of New Mexico, 1999. — 113 pp.

123. *Hornik, K.* Multilayer feedforward networks are universal approximators / K. Hornik, M. Stinchcombe, H. White // Neural networks. — 1989. — Vol. 2, no. 5. — Pp. 359–366.

124. *Hsu, C.-W.* A comparison of methods for multiclass support vector machines / C.-W. Hsu, C.-J. Lin // IEEE transactions on Neural Networks. — 2002. — Vol. 13, no. 2. — Pp. 415–425.

125. *Ilgun, K.* State transition analysis: A rule-based intrusion detection approach / K. Ilgun, R. A. Kemmerer, P. A. Porras // IEEE transactions on software engineering. — 1995. — Vol. 21, no. 3. — Pp. 181–199.

126. *Jang, J.-S.* ANFIS: adaptive-network-based fuzzy inference system / J.-S. Jang // IEEE transactions on systems, man, and cybernetics. — 1993. — Vol. 23, no. 3. — Pp. 665–685.

127. *Jerne, N. K.* Towards a network theory of the immune system / N. K. Jerne // Ann. Immunol. — 1974. — Vol. 125. — Pp. 373–389.

128. *Jiang, H.* The application of genetic neural network in network intrusion detection / H. Jiang, J. Ruan // Jornal of Computers. — 2009. — Dec. — Vol. 4, no. 12. — Pp. 1223–1230.

129. *Jolliffe, I. T.* Principal Component Analysis / I. T. Jolliffe. — Springer, 2002. — 488 pp.

130. *Jyothsna, V.* A review of anomaly based intrusion detection systems / V. Jyothsna, V. R. Prasad, K. M. Prasad // International Journal of Computer Applications. — 2011. — Vol. 28, no. 7. — Pp. 26–35.

131. *Kaufman, M.* Towards a logical analysis of the immune response / M. Kaufman, J. Urbain, R. Thomas // Journal of theoretical biology. — 1985. — Vol. 114, no. 4. — Pp. 527–561.

132. *Kaynak, O.* Computational intelligence: Soft computing and fuzzy-neuro integration with applications / O. Kaynak, L. A. Zadeh, B. Türksen, I. J. Rudas. Vol. 162. — Springer, 1998. — 542 pp.

133. *Kim, J.* The artificial immune model for network intrusion detection / J. Kim, P. Bentley // 7th European congress on intelligent techniques and soft computing (EUFIT'99). Vol. 158. — 1999.

134. *Kim, S. S.* Statistical techniques for detecting traffic anomalies through packet header data / S. S. Kim, A. Reddy // IEEE/ACM Transactions on Networking (TON). — 2008. — Vol. 16, no. 3. — Pp. 562–575.

135. *Knight, T.* A multi-layered immune inspired approach to data mining / T. Knight, J. Timmis // In Proceedings of the 4th International Conference on Recent Advances in Soft Computing. — Nottingham, UK. 2002. — Pp. 266–271.

136. *Komar, M.* Development of neural network immune detectors for computer attacks recognition and classification / M. Komar, V. Golovko, A. Sachenko, S. Bezobrazov // In Proceedings of the 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS). Vol. 2. — IEEE. 2013. — Pp. 665–668.

137. *Konar, A.* Computational intelligence: principles, techniques and applications / A. Konar. — 2005. — 708 pp.

138. *Koza, J. R.* Genetic programming: on the programming of computers by means of natural selection / J. R. Koza. Vol. 1. — MIT press, 1992. — 819 pp.

139. *Krause, A.* Foundations of GTK+ development / A. Krause. — Apress, 2007. — 630 pp.

140. *Kruegel, C.* Using decision trees to improve signature-based intrusion detection / C. Kruegel, T. Toth // In Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection. — Springer. 2003. — Pp. 173–191.

141. *Kuhn, H. W.* Nonlinear programming / H. W. Kuhn, A. W. Tucker // In Proceedings of 2nd Berkeley Symposium (University of California Press). — 1951. — Pp. 481–492.

142. *Kumar, S.* A Pattern Matching Model for Misuse Intrusion Detection / S. Kumar, E. H. Spafford // In Proceedings of the 17th National Computer Security Conference. — 1994. — Pp. 11–21.

143. *Kumar, S.* A software architecture to support misuse intrusion detection / S. Kumar, E. H. Spafford // In Proceedings of the 18th National Information Security Conference. — 1995. — Pp. 194–204.

144. *Kuncheva, L. I.* Combining pattern classifiers: methods and algorithms / L. I. Kuncheva. — John Wiley & Sons, 2004. — 350 pp.

145. *Levenberg, K.* A method for the solution of certain non-linear problems in least squares / K. Levenberg // Quarterly of applied mathematics. — 1944. — Vol. 2, no. 2. — Pp. 164–168.

146. *Li, H.* Research on intelligent intrusion prevention system based on snort / H. Li, D. Liu // In Proceedings of the 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE). T. 1. — IEEE. 2010. — C. 251—253.

147. *Lin, C.-H.* Accelerating string matching using multi-threaded algorithm on GPU / C.-H. Lin, S.-Y. Tsai, C.-H. Liu, S.-C. Chang, J.-M. Shyu // In Proceedings of Global Telecommunications Conference (GLOBECOM 2010). — IEEE. 2010. — Pp. 1–5.

148. *Lorena, A. C.* A review on the combination of binary classifiers in multiclass problems / A. C. Lorena, A. C. De Carvalho, J. M. Gama // Artificial Intelligence Review. — 2008. — Vol. 30, no. 1. — Pp. 19–37.

149. *Lunt, T. F.* [et al.] A real-time intrusion-detection expert system (IDES) / T. F. Lunt, A. Tamaru, F. Gillham. — SRI International. Computer Science Laboratory, 1992. — 157 pp.

150. *Mahoney, M.* An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection / M. Mahoney, P. Chan // In Proceedings of the 6th International Symposium on Recent advances in intrusion detection (RAID 2003). — Springer. 2003. — Pp. 220–237.

151. *Marks, R. J.* Intelligence: Computational versus artificial / R. J. Marks // IEEE Transactions on Neural Networks. — 1993. — Vol. 4, no. 5. — Pp. 737–739.

152. *Marquardt, D. W.* An algorithm for least-squares estimation of nonlinear parameters / D. W. Marquardt // Journal of the society for Industrial and Applied Mathematics. — 1963. — Vol. 11, no. 2. — Pp. 431–441.

153. *Matzinger, P.* Tolerance, danger, and the extended family / P. Matzinger // Annual review of immunology. — 1994. — Vol. 12, no. 1. — Pp. 991–1045.

154. *McHugh, J.* Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory / J. McHugh // ACM Transactions on Information and System Security (TISSEC). — 2000. — Vol. 3, no. 4. — Pp. 262–294.

155. *Mehra, P.* A brief study and comparison of snort and bro open source network intrusion detection systems / P. Mehra // International Journal of Advanced Research in Computer and Communication Engineering. — 2012. — Vol. 1, no. 6. — Pp. 383–386.

156. *Miller, I.* RFC 3128. Protection Against a Variant of the Tiny Fragment Attack / I. Miller [Электронный ресурс]. — 2001. — URL: https://www.ietf.org/rfc/rfc3128.txt (visited on 08/01/2017).

157. *Moré, J. J.* The Levenberg-Marquardt algorithm: implementation and theory / J. J. Moré // In Proceedings of the Biennial Conference Held at Dundee (Lecture Notes in mathematics, Numerical analysis). — Springer, June 1978. — Pp. 105–116.

158. *Moya, M. A. C.* Analysis and evaluation of the Snort and Bro network intrusion detection systems / M. A. C. Moya: Bachelor's Thesis / Moya, Miguel A. Calvo. — Universidad Pontificia Comillas, 2008. — 80 pp.

159. *Mukkamala, S.* Intrusion detection using an ensemble of intelligent paradigms / S. Mukkamala, A. H. Sung, A. Abraham // Journal of network and computer applications. — 2005. — Vol. 28, no. 2. — Pp. 167–182.

160. *Mukkamala, S.* Intrusion detection using ensemble of soft computing paradigms / S. Mukkamala, A. H. Sung, A. Abraham // Intelligent systems design and applications. — Springer, 2003. — Pp. 239–248.

161. *Müller, K.-R.* Predicting time series with support vector machines / K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, V. Vapnik // In Proceedings of International Conference on Artificial Neural Networks. — Springer. 1997. — Pp. 999–1004.

162. *Nejad, F. S.* Structural TLR algorithm for anomaly detection based on Danger Theory / F. S. Nejad, R. Salkhi, R. Azmi, B. Pishgoo // In Proceedings of the 9th International ISC Conference on Information Security and Cryptology. — IEEE. 2012. — Pp. 156–161.

163. *Nilsson, N. J.* Artificial intelligence: a new synthesis / N. J. Nilsson. — Morgan Kaufmann Publishers, 1998. — 513 pp.

164. ntop. Accelerating Snort, Bro and Suricata with PF_RING ZC [Электронный ресурс]. — URL: http://www.ntop.org/pf_ring/accelerating-snort-bro-and-suricata-with-pf_ring-zc/ (visited on 08/01/2017).

165. ntopng. High-Speed Web-based Traffic Analysis and Flow Collection [Электронный ресурс]. — URL: https://raw.githubusercontent.com/ntop/ntopng/dev/doc/UserGuide.pdf (visited on 08/01/2017).

166. Open Information Security Foundation. Suricata. Packet Pipeline [Электронный ресурс]. — URL: https: //redmine.openinfosecfoundation.org/projects/suricata/wiki/Packet_Pipeline (visited on 08/01/2017).

167. OpenNET. Доступна система обнаружения атак Suricata 3.0 [Электронный ресурс]. — URL: https://www.opennet.ru/opennews/art.shtml?num=43773 (дата обр. 01.08.2017).

168. OSSEC 2.8.1 documentation. Man pages [Электронный ресурс]. — URL: http://ossec-docs.readthedocs.org/en/latest/programs/ (visited on 08/01/2017).

169. *Owen, J. A.* Kuby immunology, Seventh edition / J. A. Owen, J. Punt, S. A. Stranford, J. P. P. — WH Freeman New York, 2013. — 692 pp.

170. *Paxson, V.* Bro: a system for detecting network intruders in real-time / V. Paxson // Computer networks. — 1999. — Vol. 31, no. 23. — Pp. 2435–2463.

171. *Pearson, K.* On lines and planes of closest fit to systems of points in space / K. Pearson // The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. — 1901. — Vol. 2, no. 11. — Pp. 559–572.

172. *Peddabachigari, S.* Modeling intrusion detection system using hybrid intelligent systems / S. Peddabachigari, A. Abraham, C. Grosan, J. Thomas // Journal of Network and Computer Applications. — 2007. — Vol. 30, no. 1. — Pp. 114–132.

173. *Perelson, A. S.* Immune network theory / A. S. Perelson // Immunological reviews. — 1989. — Vol. 110, no. 1. — Pp. 5–36.

174. *Perelson, A. S.* Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self–non-self discrimination / A. S. Perelson, G. F. Oster // Journal of theoretical biology. — 1979. — Vol. 81, no. 4. — Pp. 645–670.

175. PF_RING UserGuide. Linux High Speed Packet Capture [Электронный ресурс]. — URL: https://github.com/ntop/PF_RING/archive/dev.zip (visited on 08/01/2017).

176. *Poole, D.* Computational intelligence: a logical approach / D. Poole, A. Mackworth, R. Goebel. — 1998. — 558 pp.

177. *Postel, J.* RFC 791. Internet protocol / J. Postel [Электронный ресурс]. — 1981. — URL: https://www.ietf.org/rfc/rfc791.txt (visited on 08/01/2017).

178. *Powers, S. T.* A hybrid artificial immune system and Self Organising Map for network intrusion detection / S. T. Powers, J. He // Information Sciences. — 2008. — Vol. 178, no. 15. — Pp. 3024–3042.

179. *Prodromidis, A.* Meta-learning in distributed data mining systems: Issues and approaches / A. Prodromidis, P. Chan, S. Stolfo // Advances in distributed and parallel knowledge discovery. — 2000. — Vol. 3. — Pp. 81–114.

180. *Ptacek, T. H.* Insertion, evasion, and denial of service: Eluding network intrusion detection / T. H. Ptacek, T. N. Newsham: tech. rep. / SECURE NETWORKS INC CALGARY ALBERTA. — Jan. 1998.

181. *Refaeilzadeh, P.* Cross-validation / P. Refaeilzadeh, L. Tang, H. Liu // Encyclopedia of database systems. — Springer, 2009. — Pp. 532–538.

182. *Riedmiller, M.* A direct adaptive method for faster backpropagation learning: The RPROP algorithm / M. Riedmiller, H. Braun // In Proceedings of IEEE International Conference on Neural Networks. — IEEE. 1993. — Pp. 586–591.

183. *Rødfoss, J. T.* Comparison of open source network intrusion detection systems / J. T. Rødfoss: MA thesis / Rødfoss, Jonas Taftø. — University of Oslo, Department of Informatics, 2011. — 85 pp.

184. *Roesch, M.* Snort users manual / M. Roesch, C. Green [Электронный ресурс]. — 2016. — URL: https://www.snort.org/documents/1 (visited on 08/01/2017).

185. *Saied, A.* Detection of known and unknown DDoS attacks using Artificial Neural Networks / A. Saied, R. E. Overill, T. Radzik // Neurocomputing. — 2016. — Vol. 172. — Pp. 385–393.

186. *Sanders, C.* Applied network security monitoring: collection, detection, and analysis / C. Sanders, J. Smith. — Elsevier, 2013. — 496 pp.

187. *Scholkopf, B.* Learning with kernels: support vector machines, regularization, optimization, and beyond / B. Scholkopf, A. J. Smola. — MIT press, 2001. — 626 pp.

188. *Selim, S.* Intrusion Detection using Multi-Stage Neural Network / S. Selim, M. Hashem, T. M. Nazmy // International Journal of Computer Science and Information Security. — 2010. — Vol. 8, no. 4. — Pp. 14–20.

189. *Sen, S.* Performance characterization & improvement of snort as an IDS / S. Sen // Bell Labs Report. — 2006.

190. *Shawe-Taylor, J.* Kernel methods for pattern analysis / J. Shawe-Taylor, N. Cristianini. — Cambridge university press, 2004. — 462 pp.

191. *Silberschatz, A.* Operating system concepts, 9th edition / A. Silberschatz, P. B. Galvin, G. Gagne. — Wiley, 2012. — 976 pp.

192. *Smith, D. J.* Immunological memory is associative / D. J. Smith, S. Forrest, A. S. Perelson // In Proceeding of International Conference on Multiagent Systems (In Workshop Notes, Workshop 4: Immunity Based Systems). — 1996. — Pp. 62–70.

193. *Sommer, R.* Outside the closed world: On using machine learning for network intrusion detection / R. Sommer, V. Paxson // In Proceedings of the IEEE Symposium on Security and Privacy. — IEEE. 2010. — Pp. 305–316.

194. *Stephen, G. A.* String searching algorithms / G. A. Stephen. Vol. 6. — World Scientific, 1994. — 256 pp.

195. *Takagi, T.* Fuzzy identification of systems and its applications to modeling and control / T. Takagi, M. Sugeno // IEEE transactions on systems, man, and cybernetics. — 1985. — No. 1. — Pp. 116–132.

196. *Tarakanov, A. O.* Immunocomputing: principles and applications / A. O. Tarakanov, V. A. Skormin, S. P. Sokolova. — Springer, 2003. — 193 pp.

197. The Bro Network Security Monitor. Load Balancing [Электронный ресурс]. — URL: https://www.bro.org/documentation/load-balancing.html (visited on 08/01/2017).

198. The Bro Network Security Monitor. Protocol Analyzers [Электронный ресурс]. — URL: https://www.bro.org/sphinx/script-reference/proto-analyzers.html (visited on 08/01/2017).

199. *Tizard, I. R.* Veterinary immunology: An introduction, 7th edition / I. R. Tizard. — Saunders, 2004. — 476 pp.

200. *Toosi, A. N.* A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers / A. N. Toosi, M. Kahani // Computer communications. — 2007. — Vol. 30, no. 10. — Pp. 2201–2212.

201. *Turing, A. M.* Computing machinery and intelligence / A. M. Turing // Mind. — 1950. — Vol. 59, no. 236. — Pp. 433–460.

202. *Vaitsekhovich, L.* Intrusion detection in TCP/IP networks using immune systems paradigm and neural network detectors / L. Vaitsekhovich, V. Golovko // XI International PhD Workshop OWD. — 2009. — Pp. 219–224.

203. *Wang, G.* A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering / G. Wang, J. Hao, J. Ma, L. Huang // Expert systems with applications. — 2010. — Vol. 37, no. 9. — Pp. 6225–6232.

204. *Werbos, P. J.* Beyond regression: New tools for prediction and analysis in the behavioral sciences / P. J. Werbos: PhD thesis / Werbos, Paul John. — Harvard University, MA, 1974.

205. *White, J. S.* Quantitative analysis of intrusion detection systems: Snort and Suricata / J. S. White, T. Fitzsimmons, J. N. Matthews // SPIE Defense Security and Sensing Cyber Security Conference. Vol. 8757. — 2013.

206. *Wu, S. X.* The use of computational intelligence in intrusion detection systems: A review / S. X. Wu, W. Banzhaf // Applied Soft Computing. — 2010. — Vol. 10, no. 1. — Pp. 1–35.

207. Хакер. Большой Брат: обзор системы обнаружения вторжений Bro [Электронный ресурс]. — URL: https://xakep.ru/2015/06/28/big-bro-197/ (дата обр. 01.08.2017).

208. *Yasm, C.* Prelude as a hybrid IDS framework / C. Yasm // SANS Institute. — 2009.

209. *Ye, N.* An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems / N. Ye, Q. Chen // Quality and Reliability Engineering International. — 2001. — Vol. 17, no. 2. — Pp. 105–112.

210. *Zaraska, K.* Prelude IDS: current state and development perspectives / K. Zaraska [Электронный ресурс]. — 2003. — URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.5542%5C&rep=rep1%5C&type=pdf.

211. *Zhou, Y.-p.* Hybrid Model Based on Artificial Immune System and PCA Neural Networks for Intrusion Detection / Y.-p. Zhou // Asia-Pacific Conference on Information Processing. Vol. 1. — IEEE. 2009. — Pp. 21–24.

212. *Zhou, Z.-H.* Ensemble methods: foundations and algorithms / Z.-H. Zhou. — CRC press, 2012. — 218 pp.

213. *Ziemba, G.* RFC 1858. Security Considerations for IP Fragment Filtering / G. Ziemba, D. Reed, P. Traina [Электронный ресурс]. — 1995. — URL: https://www.ietf.org/rfc/rfc1858.txt (visited on 08/01/2017).

Алфавитно-предметный указатель

A

Б

Г

Д

И

К

М

Н

П

С

Ф

Э

Список рисунков

Список таблиц

Список алгоритмов

Приложение А

Исследование открытых сигнатурных СОА

А.1 Общее представление об исследуемых СОА

В эпоху цифровых технологий одной из ключевых проблем в современном обществе является защита информации. Это приводит к созданию определенных систем и средств, которые должны иметь в своем наличии на сетевом уровне некоторые механизмы защиты и противодействия хакерским угрозам. В настоящее время считается важным обеспечить наибольший уровень защищенности в корпоративных сетях любой организации, которая хранит критически важные данные. Успех и конкурентоспособность такой организации во многом зависят именно от использования эффективных средств, удовлетворяющих условиям выше. Одним из наиболее известных и хорошо зарекомендовавших себя средств для решения этой задачи являются СОА. По способу сбора информации выделяют хостовые и сетевые СОА. В данном приложении больший акцент сделан в сторону исследования и критического анализа тех СОА, которые функционируют именно на сетевом уровне. Такие СОА можно определить как программные или аппаратные устройства, предназначенные для выявления попыток несанкционированного доступа, фактов нарушения безопасности в рамках сетевого взаимодействия между хостами и любых других аномальных действий в компьютерной сети.

Для обзора были выбраны пять СОА, которые имеют открытый программный код и распространяются по бесплатным лицензиям Gnu GPL и BSD. Среди них представлены Snort, Suricata, Bro, OSSEC, Prelude. Анализ данных СОА проводился на уровне изучения их исходных текстов, руководств пользователя, программной документации и прочих источников технической и научной литературы, находящихся в открытом доступе.

В таблице 11 приведены общие сведения о рассматриваемых СОА. Среди них были выделены следующие индикаторы: автор, компания-разработчик или сопровождающее сообщество, сайт производителя, сайт системы, основной АЯВУ для разработки системы, тип лицензии для распространения системы, версия исследуемой СОА, год начала разработки или выпуска первого бета-релиза, общее число компилируемых файлов (с расширениями *.c, *.C, *.cc, *.cpp, *.cxx, *.ec, *.h, *.H, *.hh, *.hpp, *.pcc, *.pgc), общее количество непустых строк кода на C/C++ без учета комментариев, размер разархивированной директории.

Таблица 11 — Общие сведения об исследуемых СОА

Характеристика	Snort	Suricata	Bro	OSSEC	Prelude
Автор	Martin Roesch	Eric Leblond	Vern Paxson	Daniel B. Cid	Yoann Vandoorselaere
Компания-разработчик или сопровождающее сообщество	Cisco, ранее Sourcefire	Open Information Security Foundation	International Computer Science Institute in Berkeley	OSSEC Github	Prelude community, ранее Edenwall
Сайт производителя	http://www.cisco.com	http://oisf.net/	https://www.icsi.berkeley.edu/icsi/	http://ossec.github.io/	https://www.prelude-siem.org/
Сайт системы	https://www.snort.org/	https://suricata-ids.org/	https://www.bro.org/	http://ossec.github.io/	https://www.prelude-siem.org/
Основной АЯВУ для разработки системы	C	C	C++	C	C
Тип лицензии для распространения системы	GNU General Public License v.2	GNU General Public License v.2	BSD License	GNU General Public License v.2	GNU General Public License v.2
Версия исследуемой СОА	2.9.8.2	3.0.1	2.4.1	2.9.0	1.2.6
Год начала разработки или выпуска первого бета-релиза	1998	2009	1995	2004	1998
Общее число компилируемых файлов (с расширениями *.c, *.C, *.cc, *.cpp, *.cxx, *.ec, *.h, *.H, *.hh, *.hpp, *.pcc, *.pgc)	1000	886	865	501	1146
Общее количество непустых строк кода на C/C++ без учета комментариев	289847	378430	334299	88093	293652
Размер разархивированной директории	25824Кб	23904Кб	90112Кб	27932Кб	49556Кб

В таблице 12 представлены некоторые исследовательские работы, в которых рассматриваются СОА, и раскрыто их основное содержание.

Таблица 12 — Статьи, связанные с исследованием СОА

Авторы, ссылка и год публикации статьи	Исследуемые СОА	Краткое содержание
D.J. Day, B.M. Burns, [85], 2011	Snort, Suricata	Исследование направлено на сравнение СОА по потреблению системных ресурсов (загрузка процессора в зависимости от пропускной способности сетевого канала, время чтения pcap-файлов в оффлайн-режиме, использование нескольких ядер CPU, выявление зависимости числа отброшенных пакетов от доступности CPU), количеству ложноположительных срабатываний и количеству верно обнаруженных атак. Suricata отличает бóльшая точность в обнаружении атак, более равномерное использование и распределение задач между ядрами процессора, ускорение обработки трафика с увеличением числа подключаемых ядер и меньшая скорость обработки pcap-файла. Snort характеризуется менее устойчивой балансировкой нагрузки между ядрами и высокой скоростью чтения сетевых дампов.
C.A.P. Boyce, A.N. Zincir-Heywood, [62], 2003	Dragon, Firestorm, Snort, Prelude	Статья включает сравнение нескольких СОА по критерию обнаружения сетевых атак на примере сетевых дампов DARPA 1999. При проведении тестов лучший результат показала коммерческая СОА Dragon.
J.S. White, T.T. Fitzsimmons, J.N. Matthews, [205], 2013	Snort, Suricata	Статья посвящена сравнительному анализу двух СОА. Эксперименты проводились для выявления закономерностей объема потребляемых ресурсов (загрузка CPU, размер потребляемой RAM) и скорости обработки пакетов в зависимости от числа используемых в системе ядер и настроек СОА. Было отмечено, что наилучшей производительностью обладает Suricata.
M. Ennert, E. Chovancová, Z. Dudláková, [98], 2015	Snort, Suricata	В статье рассмотрено несколько экспериментов, направленных на тестирование СОА. Среди таких тестов представлены следующие: определение скорости обработки pcap-файла, генерация атак с целью определения точности их обнаружения, определение потребления системных ресурсов каждой из СОА, определение точности обнаружения атак с фоновым „белым“ трафиком, использование множества запущенных СОА. Snort показала более экономное расходование системных ресурсов. Suricata продемонстрировала бóльшую точность в обнаружении атак и меньшее время, потраченное на чтение pcap-файла.

Продолжение таблицы 12

Авторы, ссылка и год публикации статьи	Исследуемые СОА	Краткое содержание
E. Albin, [49], 2011	Snort, Suricata	В работе проводится сравнение двух СОА по результатам их использования в следующих экспериментах: измерение производительности СОА на реальном трафике и на статичном pcap-файле (скорость чтения пакетов) с учетом затраченных на обработку пакетов системных ресурсов (использование ресурсов CPU и RAM), определение показателя верного обнаружения атак. По результатам исследования установлено, что Suricata показывает более высокую скорость обработки трафика за счет многопоточной архитектуры и, как следствие, требует в процессе своего функционирования большее количество системных ресурсов (память и процессорное время). Пропускная способность Snort ограничена отметкой в 200–300 Mb/sec, и процедура использования нескольких экземпляров Snort является проигрышной по сравнению с запуском одной копии Suricata.
M.A.C. Moya, [158], 2008	Snort, Bro	Исследование представляет собой обзор двух СОА. Описывается внутреннее устройство каждой из систем, приводятся правила для обнаружения атак. Выявлено несколько отличительных особенностей при анализе сетевого трафика данными СОА. Отмечено, что Snort работает на пакетном уровне и производит анализ как отдельных сырых IP-фрагментов, так и восстановленного сетевого потока, в то время как анализаторы Bro запускаются только после выполнения полной дефрагментации пакетов.
J.T. Rødfoss, [183], 2011	Snort, Suricata, Bro	Целью исследования является сравнение показателей производительности рассматриваемых СОА. Тесты производились с целью определения времени обработки входного pcap-файла, подсчета числа сгенерированных предупреждений в процессе его чтения, а также для обнаружения атак в трафике, сгенерированном при помощи инструмента Metasploit.

Данное приложение структурировано следующим образом. Вначале (в разделах А.2, А.3, А.4, А.5, А.6) анализируются рассматриваемые СОА Snort, Suricata, Bro, OSSEC, Prelude. В разделе А.7 сравниваются характеристики этих СОА. Раздел А.8 посвящен рассмотрению способов улучшения функционирования СОА. В разделе А.9 исследуется способность сетевых СОА к обнаружению атак со скрытием и со вставкой, а в разделе А.10 — устойчивость СОА к стрессовым сетевым нагрузкам.

A.2 Snort

Snort является одним из самых популярных open-source инструментов в арсенале системных администраторов при решении множества проблем, связанных с обеспечением безопасности компьютерных хостов на уровне сети. Первостепенной задачей данного ПО является выявление нарушений в политиках сетевой безопасности, что соответствует режиму обнаружения сетевых атак. Кроме того, согласно руководству пользователя [184] данная система способна функционировать как анализатор и регистратор пакетов. В этих режимах система соответственно отображает данные в окно вывода (терминал) и записывает данные на диск в виде лог-файлов. Сама система разрабатывалась с учетом требований к высокому быстродействию, в первую очередь ставшему возможным благодаря простоте ее ядра. Ядро системы построено на основе сравнения по правилам и в большинстве случаев использует быстрый поиск подстроки с применением алгоритма Бойера—Мура и механизма регулярных выражений на основе библиотеки libpcre. На рисунке А.1 схематически представлены базовые компоненты системы Snort и основные функции, вызываемые внутри каждого компонента и при передаче управления между ними [52, 56, 155, 189].

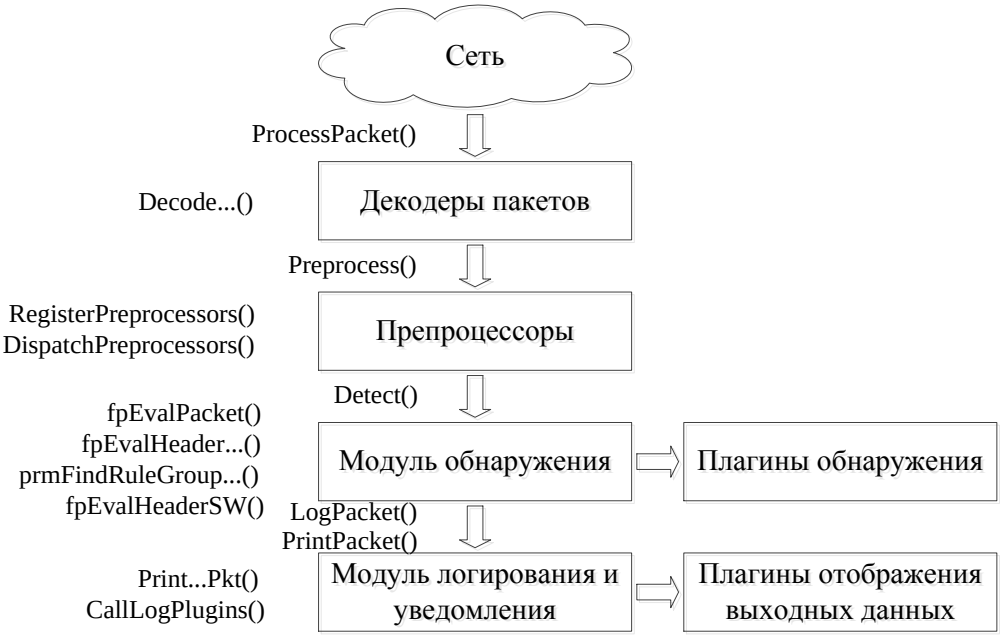


Рисунок А.1 — Компоненты Snort

В состав системы входят четыре компонента: различные декодеры пакетов, препроцессоры, модуль обнаружения, а также модуль логирования и уведомления об обнаруженных атаках. Взаимодействие между компонентами осуществляется в рамках единого адресного пространства процесса snort (с возможностью подгрузки динамически исполняемых модулей) и происходит посредством прямых вызовов функций, которые приводят к непосредственному запуску очередного компонента.

Для каждого захваченного из сети пакета вызывается функция ProcessPacket, которая приводит к запуску декодера (grinder) полученного пакета. В данный момент Snort версии 2.9.8.2 поддерживает такие канальные протоколы, как Ethernet, loopback, raw IP, raw IPv4, raw IPv6, PPP, IEEE 802.11, IEEE 802.5 Token Ring, FDDI, Cisco HDLC, SLIP, PPP over serial with HDLC encapsulation, Linux cooked sockets, OpenBSD PF log, old OpenBSD PF log. Для каждого из этих и более высокоуровневых протоколов определены соответствующие декодеры пакетов, представляющие собой простые в реализации модули, целью которых является последовательный разбор закрепленных за ними уровней внутри структуры пакета с заполнением соответствующих полей данными перехваченного пакета. Основой для реализации этих модулей является библиотека libpcap (winpcap), предоставляющая весь необходимый набор функций для чтения и анализа пакетов. Большей частью функционирование декодеров после захвата пакета сводится к наложению на структуру пакета определенной части поступивших данных с автоматическим заполнением необходимых полей при помощи вызова системной функции memcpy и инвертирования байтового потока в хостовый порядок при помощи вызовов ntohs/ntohl. В процессе декодирования обрабатываемых пакетов каждым модулем расставляются указатели на границы смены соседних блоков внутри каждого пакета. Поэтому по завершении работы одного декодера можно с легкостью возобновить работу следующего декодера, ответственного за инициализацию полей соответствующего протокола в рамках нового участка пакета. С этой целью в систему Snort жестко запрограммирована четкая последовательность вызывающих-

ся процедур при обработке типовых пакетов. Так, для декодирования Ethernet-фреймов предусмотрена функция DecodeEthPkt, которая, в свою очередь, в зависимости от значения поля ether_type внутри Ethernet-кадра передает управление одной из функций DecodeIP (ether_type = 2048), DecodeARP (ARP: ether_type = 2054, RARP: ether_type = 32821), DecodeVLAN (ether_type = 33024), DecodeIPV6 (ether_type = 34525), DecodeEthLoopback (ether_type = 36864), DecodeIPX (ether_-type = 33079) и т.д. Для декодирования вложенных в IP-пакет данных сперва вызывается функция DecodeIPOptions для разбора IP-опций (при условии их наличия), следом за которой вызывается одна из функций DecodeTCP, DecodeUDP, DecodeICMP, DecodeIPV6, DecodeGRE, DecodeESP, DecodeAH и т.д. для соответствующих значений поля protocol 6, 17, 1, 41, 47, 50, 51. Следующий этап работы системы Snort — запуск препроцессоров. С этой целью вызывается функция Preprocess (на самом деле эта функция также приводит к запуску модуля обнаружения атак). Препроцессоры по своему назначению напоминают декодеры, но в отличие от них способны производить расширенный набор более сложных операций, таких как нормализация данных, склейка пакетов в один поток, сохранение состояний для управления сессией. Функция RegisterPreprocessors заносит в однонаправленный список модули с первоначальной их инициализацией. Так, по умолчанию системой загружаются модули, выполняющие следующие функции: декодирование ARP-пакетов для обнаружения атак ARPspoof с подменой аппаратных адресов (SetupARPspoof), нормализация сетевых и транспортных протоколов IP, ICMP, IPv6, ICMPv6, TCP (SetupNormalizer), дефрагментация IP-пакетов (SetupFrag3), отслеживание сессий и сохранение их состояний для протоколов IP, TCP, UDP, ICMP (SetupSessionManager), реассемблирование содержимого TCP-потоков с учетом непоследовательности поступления пакетов (SetupStream6), сбор нескольких RPC фрагментированных пакетов в единую запись (SetupRPCDecode), обнаружение атаки back orifice (SetupBO), анализ HTTP-заголовков (SetupHTTPInspect), измерение теоретической максимальной производительности Snort (SetupPerfMonitor), обнаружение атак, направленных на сканирование портов и сбор информации об уязвимостях ска-

нируемой системы (SetupSfPortscan). Также среди дополнительных препроцессоров можно назвать декодирование команд для сетевых служб SMTP, POP3, IMAP, FTP, Telnet, SSH, DNS, SSL/TLS, SIP и т.д. Функция DispatchPreprocessors отвечает за вызов каждого из таких препроцессоров со своим набором опций, которые указаны в конфигурационном файле /etc/snort/snort.conf. После работы препроцессоров функция Preprocess вызывает функцию Detect при условии, что Snort запущен в режиме обнаружения сетевых атак. К каждому из пакетов типа TCP, UDP, ICMP, IP применяется набор правил соответствующего типа посредством вызова функций fpEvalHeaderTCP, fpEvalHeaderUDP, fpEvalHeaderICMP, fpEvalHeaderIP. Этим наборам правил назначается групповое правило (prmFindRuleGroup) для обработки пакетов наиболее общего вида соответствующего протокола. Наконец, последовательность вызовов функций fpEvalHeaderSW приводит к обходу иерархического списка вложенных правил. Модули вывода (логирования и уведомления) предназначены для приведения выходных данных к унифицированному виду. Выходные данные могут генерироваться, к примеру, во время обработки событий, посланных одним из модулей обнаружения при срабатывании одного из правил. Среди поддерживаемых форматов вывода можно назвать формат базы данных, XML, tcpdump, syslog.

Система Snort является одним из лучших программных решений в области сетевой безопасности, которое сочетает в себе легковесность, модульную расширяемость и прозрачную архитектуру. Кроме того, в системе реализовано множество полезных дополнительных функций по анализу сетевого потока, таких как работа в режиме оффлайн с чтением сохраненных на диске сетевых дампов, вычисление статистических сведений о перехваченных пакетах с группировкой по конкретным протоколам, подсчет числа отфильтрованных и отброшенных пакетов, отображение затраченных в процессе анализа трафика временных и аппаратных ресурсов. Среди достоинств системы также стоит подчеркнуть наличие в открытом доступе большого числа готовых к практическому применению правил, которые предоставляются активно разрабатывающим данную систему сообществом.

A.3 Suricata

Suricata проектировалась с расчетом на использование в оборудовании, в котором реализована поддержка современных технологий в области аппаратного обеспечения. Разработчики системы особое внимание уделили тому факту, что система будет запускаться именно на многопроцессорных/многоядерных машинах. Также для достижения этой цели были сделаны попытки максимально оптимизировать функционирование разрабатываемой системы с ориентацией на современные модели чипсетов CPU и GPU. Была специально разработана многопоточная архитектура, которая позволила использовать весь вычислительный потенциал и мощь этих машин в приемлемой степени.

Функционально архитектура системы включает несколько строительных блоков: потоки, модули и очереди. Работа этих блоков объединяется конвейерным принципом. Согласно этому подходу каждый из модулей способен работать независимо от остальных, принимая на каждом такте данные, отличные от обрабатываемых в этот же момент другими модулями. Осуществление такого дробления приводит к тому, что несколько различных пакетов могут обрабатываться системой одновременно. Отметим, что основные этапы функционирования Suricata при обработке пакетов такие же, как у Snort. Отличием является лишь то, как было указано выше, что этапы выполняются асинхронно. С этой целью взаимодействие компонентов осуществляется через механизм буферных очередей. Одним из режимов работы системы, который применяется в системе по умолчанию, называется autofp и соответствует рисунку А.2 [166], является последовательный механизм обработки и передачи пакетов с начальной инициализацией нескольких потоков для каждого модуля. Так, для этапов 1, 2, 3 и 5 создается по одному потоку, для выполнения всех этих потоков выделяется одно ядро. Остальные имеющиеся в наличии ядра распределяются по одному для работы каждого потока, выполняющего функции модуля обнаружения (см. файл runmode-pcap-file.c).

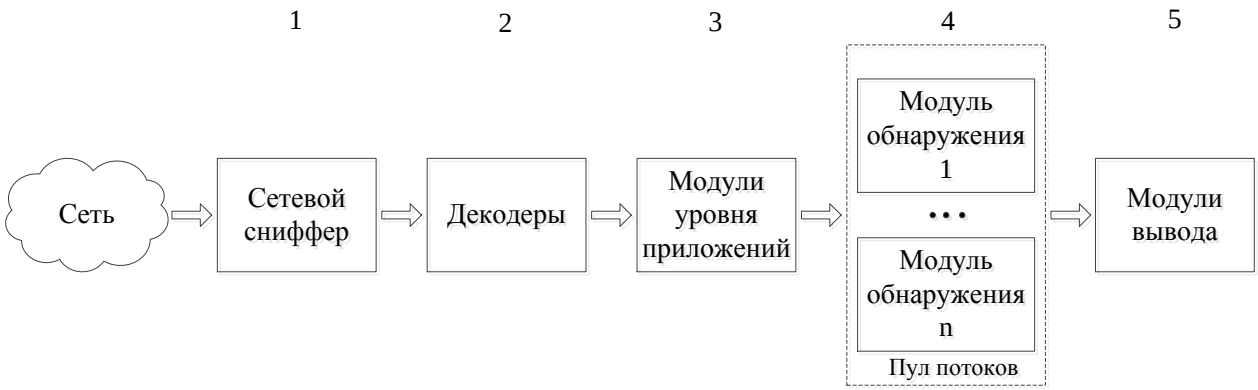


Рисунок А.2 — Компоненты Suricata

Среди отличительных особенностей системы стоит выделить следующие:

– расширяемость за счет возможности написания скриптовых правил на АЯВУ Lua [167];
– ускорение вычислений за счет возможности частичного переноса исполняемого кода на графические процессоры с поддержкой технологий CUDA/OpenCL (в частности, вычисление регулярных выражений осуществляется на GPU при условии конфигурирования и последующей компиляции системы с флагом --enable-cuda);
– многопоточность;
– отсутствие привязки к номеру сетевого порта для распознавания типа прикладного сервиса за счет наличия в системе встроенного DPI-парсера [167];
– наличие мощного анализатора HTTP-контента на основе библиотеки libhtp [167].

Недостатками системы являются довольно скудная документация и частичная поддержка правил Snort (обрабатывается только ограниченный набор команд).

Данная система, в первую очередь, позиционируется авторами как система предотвращения атак. Акцент в эту сторону обусловлен наличием в системе функций блокировки трафика при помощи библиотек libnetfilter-queue и libnfnetlink.

A.4 Bro

Bro зарождалась как исследовательский прототип в недрах международного института компьютерных наук в Беркли (International Computer Science Institute in Berkeley). Данная СОА предназначена для работы в высоконагруженных компьютерных сетях, ориентирована на оптимизацию обработки сетевых событий, возникающих в процессе анализа сетевого трафика. Механизм, применяемый в подобной системе, называется событийно-ориентированным анализом, который заключается в определении пользователем функций ответной реакции (callback-functions) на возникновение в системе определенных сетевых событий. Такой подход имеет ряд преимуществ по сравнению с традиционными списками сигнатур: (1) позволяет интегрировать в систему собственные модули для обработки интересующих событий без необходимости их компиляции (в отличие от модулей Snort) и пересборки всей системы целиком; (2) предоставляет высокоуровневый интерфейс для удобного мониторинга (отслеживания) изменения указанных полей сетевых пакетов и параметров сессий; (3) обеспечивает простой доступ к вызову внешних команд Unix-оболочки.

Среди особенностей системы стоит отметить следующие [207]:

– расширяемость.

В основу работы системы заложена возможность добавления пользовательских правил для анализа сетевого потока. Подобные правила описываются в виде скриптовых сценариев на внутреннем языке системы Bro, синтаксис и семантика которого представляют собой смесь языков C и Python. В рамках данного языка в пользование программистов предоставляется набор встроенных в ядро языка глобальных структур данных. Переменные этих типов данных выступают в роли фактических параметров при вызове функций-обработчиков событий, которые генерируются во время прослушивания сетевого интерфейса. Кроме того, существует возможность для конвертации правил Snort во внутреннее представление системы Bro при помощи Python-скрипта snort2bro, хотя поддержка

некоторых последних функций Snort по-прежнему ограничена [119]. В общем, расширение основного функционала системы построено вокруг т.н. анализаторов событий, представление о которых дано ниже.

– эффективность.

Система изначально разрабатывалась как программный продукт, который будет запускаться для анализа сетевых пакетов в высокоскоростных магистральных каналах передачи данных. С этой целью была разработана специальная архитектура системы, описание которой представлено ниже.

– возможность анализа протоколов различного уровня.

В системе реализована поддержка множества протоколов от канального до прикладного уровней модели OSI. Среди них представлены основные протоколы прикладного уровня HTTP, FTP, SMTP, DNS, SSH, DHCP, транспортного уровня TCP, UDP, сетевого уровня ICMP, канального уровня ARP и пр.

Система Bro в свой состав включает следующие компоненты: подсистему перехвата трафика, подсистему генерации и обработки событий и интерпретатор скриптов (рисунок А.3) [170].

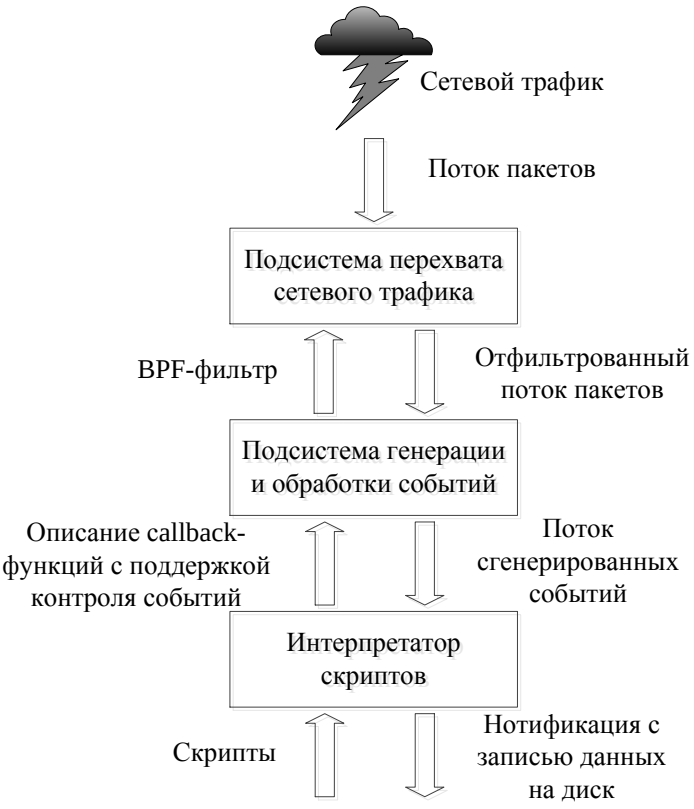


Рисунок А.3 — Компоненты Bro

Подсистема перехвата сетевого трафика представлена в виде сниффера сетевых пакетов, который реализован с использованием библиотеки libpcap. На этот уровень приходится наибольший объем данных, поэтому требования, предъявляемые к этому компоненту, заключаются в необходимости быстрого разбора сетевого трафика и фильтрации только заведомо нужных пакетов. К примеру, можно настроить систему для игнорирования пакетов IPv6, ARP, RARP и пр. и приема только пакетов с типом заголовка сетевого уровня IPv4 с фильтрацией по TCP-портам 21 (FTP), 22 (SSH), 143 (IMAP). BPF-фильтр в этом случае будет выглядеть следующим образом: ether proto 0x0800 and ip proto 6 and (tcp port 21 or tcp port 22 or tcp port 143). Посредством применения подобных правил удается снизить поток соединений на уровне ядра ОС и т.с. избежать необходимости копирования бинарных данных в пространство пользователя и дополнительного переключения контекста между кольцами защиты ОС, повышая общую производительность системы без пропуска пакетов. Подсистема генерации и обработки событий представляет собой ключевой компонент системы Bro, который позволяет выполнять указанные в callback-функциях действия, соответствующие событиям внутри анализаторов протоколов, и контролировать генерацию событий. Анализаторы протоколов являются неотъемлемыми составляющими данного модуля, которые в процессе своей работы порождают различные события, являющиеся важными с точки зрения функционирования этого протокола. Например, для протокола TCP анализатор ANALYZER_TCP генерирует такие события, как успешное установление соединения (connection_established), неуспешная попытка установления соединения (connection_attempt), сброс соединения посредством отправки/приема RST-пакета (connection_rejected/connection_reset), нормальное завершение сессии (connection_finished), отправка/прием TCP-пакета (tcp_packet) и пр., для протокола POP3 анализатор ANALYZER_POP3 генерирует такие события, как запрос на установление сессии (pop3_request), ответ сервера на установление сессии (pop3_reply), отправка данных сервером в ответ на команды RETR и TOP (pop3_data), нарушение перехода между состояниями в сессии (pop3_unexpected), TLS-шифрование (pop3_starttls), успешная

(pop3_login_success) и неуспешная аутентификация клиента (pop3_login_failure), для протокола RADIUS анализатор ANALYZER_RADIUS генерирует такие события как отправка/прием сообщения (radius_message), отправка/прием сообщения, содержащего атрибуты протокола (radius_attribute) [198]. В результате задания содержимого callback-функций, привязанных к определенным событиям, анализаторы протоколов способны осуществлять целенаправленные действия по анализу сетевого трафика и выявлению в нем вредоносной активности в соответствии с набором команд, указанных программистом. Третий компонент системы Bro — это интерпретатор скриптов. Предназначение данного модуля заключается в синтаксическом анализе содержимого пользовательских и системных сценариев, проверке их корректности и построении абстрактного синтаксического дерева. Сценарии описывают различные вспомогательные функции, а также обработчики событий, идентичные им, но не имеющие возвращаемого значения. Каждое событие, передаваемое в интерпретатор, поступает с частично скомпилированным представлением соответствующего ему обработчика. Интерпретатор связывает значения фактических параметров внутри поступающих событий с формальными параметрами функции-обработчика. Скомпилированный код может выполнять такие команды, как регистрацию уведомлений в режиме реального времени, запись данных на диск, создание новых событий, изменение внутреннего состояния для доступа и контроля к вызывающимся впоследствии обработчикам событий. Среди недостатков системы можно отметить, что наличие высокоуровневых функций анализа сетевого трафика может приводить к пропуску нестандартных низкоуровневых видов атак (к примеру, скрытых атак, атак, связанных с некорректно вычисленной контрольной суммой или с некорректными IP-опциями для отдельных пакетов). Кроме того, система не рассчитана для запуска „из коробки“ и требует от операторов и администраторов предварительной грамотной настройки и глубоких знаний в области межсетевого взаимодействия и функционирования сетевых протоколов. Немаловажный минус этой СОА заключается в отсутствии каких-либо графических инструментов для просмотра записей и возможности их сохранения в БД.

A.5 OSSEC

OSSEC является масштабируемой хостовой СОА, которая выполняет ряд следующих функций [116]: (1) анализ целостности файлов; (2) проверка системного реестра Windows; (3) обнаружение руткитов, попыток повышения привилегий до уровня суперпользователя и выполнения команд от его имени (только для UNIX); (4) генерация предупреждений в режиме реального времени; (5) обнаружение ошибок сегментации (segfaults) в приложениях и других критических событий в ядре ОС; (6) активная ответная реакция на события различного рода.

На рисунке А.4 представлена схема функционирования OSSEC [80].

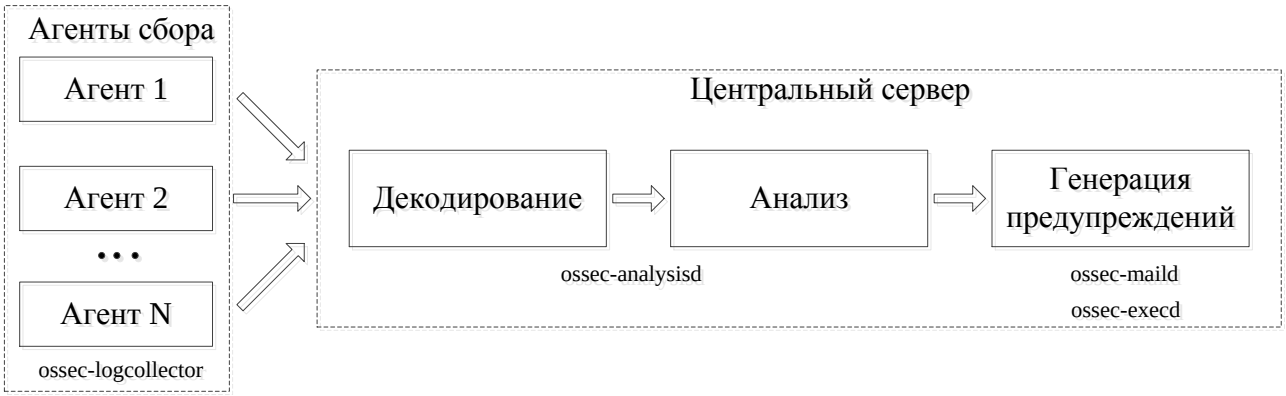


Рисунок А.4 — Схема функционирования OSSEC

Основными этапами работы системы являются сбор данных агентами, декодирование полученной от них информации, анализ и генерация предупреждений в соответствии со списком установленных правил. На клиентской стороне агенты отслеживают изменения лог-файлов и отправляют их содержимое в архивированном виде центральному серверу с опциональным применением шифрования. Серверный компонент, в свою очередь, производит прием, распаковку, декодирование и анализ данных и в случае срабатывания одного из правил оповещает администратора системы посредством отправки предупреждения через электронную почту или выполняет активное противодействие. При старте системы запускаются следующие демоны [168]: (1) ossec-authd (процесс регистрации агентов для подключения к серверу); (2) ossec-analysisd (процесс анализа

лог-файлов в соответствии с заданными правилами); (3) ossec-csyslogd (процесс отправки предупреждений через syslog); (4) ossec-remoted (процесс взаимодействия сервера с клиентскими агентами; по умолчанию для прослушивания входящих от агентов соединений используется UDP-порт 1514); (5) ossec-logcollector (процесс мониторинга изменения контролируемых файлов и вывода сообщений системных команд); (6) ossec-agentd (процесс отправки лог-файлов на серверную сторону); (7) ossec-maild (процесс отправки e-mail-уведомлений); (8) ossec-execd (процесс выполнения активных действий: перезапуск компонентов, блокировка подозрительных сетевых соединений); (9) ossec-monitord (процесс отслеживания подключений к агентам и сжатия дневных лог-файлов); (10) ossec-syscheckd (процесс проверки изменения у файлов контрольных сумм, прав доступа и владельцев); (11) ossec-dbd (процесс вставки записей с предупреждениями в БД). OSSEC имеет ряд недостатков, которые, впрочем, свойственны всяким хостовым СОА: (1) обнаруживается лишь постфактум проведения само́й атаки; (2) отсутствуют превентивные меры по ликвидации и защите от первых проявлений сетевых атак (как следствие из первого пункта); (3) отслеживаются лишь те системные и сетевые службы, которые ведут запись результатов своей работы в виде лог-файлов, обрабатываемых данной СОА. Среди преимуществ OSSEC как и любой хостовой СОА можно назвать: (1) низкий уровень ложных срабатываний; (2) возможность обнаружения атак со скрытием и со вставкой, которые не обнаруживаются сетевыми СОА ввиду специфичности реализации сетевого стека на защищаемом хосте; (3) способность анализа контента, передаваемого в зашифрованном виде по сети.

Система обладает мощным механизмом анализа лог-файлов. Язык настройки системы представлен в виде XML-разметки и позволяет создавать цепочки зависимостей правил и декодеров при помощи тегов if_sid и parent соответственно. Также на уровне декодеров при помощи тегов regex и order можно анализировать и получать отдельные поля внутри лог-записей, которые могут быть использованы для формирования новых тегов на уровне правил и сопоставления найденных полей с заданным значениями.

A.6 Prelude

Prelude является универсальной системой, совмещающей в себе основные и необходимые функции по управлению информационной безопасностью. В их число входят следующие важные операции [208]:

– сбор данных.

Prelude отслеживает данные о событиях безопасности как на уровне хоста, так и на уровне сети. С этой целью к системе подключаются сенсоры, которые выполняют функции по перехвату и анализу сетевого трафика, мониторингу изменения конфигурационных файлов и их хеш-сумм (контролю целостности), проверке содержимого лог-файлов на наличие подозрительных записей, оповещению о выходе из строя какого-либо устройства или системной/сетевой службы. Логически такие датчики представляются как СОА, модули ФС и внешних устройств, анализаторы журналов аудита.

– нормализация.

Выходные события от компонентов Prelude представляются в виде IDMEF-сообщений, формат которых есть XML-подобный открытый стандарт для обмена данными между устройствами системы [90]. Данные сохраняются в единое централизованное хранилище, записи которого согласно IDMEF-формату жестко структурированы и имеют унифицированное представление. Это позволяет сопровождающим разработчикам избавиться от написания множества декодирующих процедур для выходных данных от каждого устройства.

– агрегация.

В системе осуществлена возможность подключения разнородных устройств от различных производителей (вендоров). Использование таких систем, как Snort, Suricata, OSSEC, Samhain, в качестве сенсоров для управления при помощи Prelude позволяет объединить результаты их работы и избежать анализа специфичных лог-файлов со стороны аналитиков.

– фильтрация.

Событиям в системе можно сопоставить определенный уровень угрозы (severity), который позволяет администраторам обратить внимание на наиболее опасные события и своевременно принять соответствующие меры.

– корреляция.

Система получает информацию об угрозах от нескольких менеджеров управления и выявляет зависимость этих данных в соответствии с заданными правилами корреляции.

– оповещение об обнаруженных угрозах.

Данная функция реализована в системе при помощи создания отчетов. Это позволяет организациям представлять данные об обнаруженных угрозах в удобном визуальном виде.

Prelude как система, обладающая всеми аспектами SIEM-мониторинга, принадлежит к классу систем, отличных от СОА. Но тем не менее в данное приложение автором она была включена по той первоочередной причине, что она располагает в своем наличии компонентами обнаружения сетевых атак и выполняет ряд функций, свойственных СОА. Кроме того, наибольший интерес представляет сам факт сравнения Prelude с остальными СОА на основе выбранных критериев, описанных в разделе А.7. В составе системы Prelude выделяются следующие программные компоненты: (1) сенсоры (как и в любой другой распределенной системе, такие датчики устанавливаются в контролируемых местах, где важен учет проходящих данных); (2) менеджеры управления (Prelude-managers; задачей этих компонент является сбор информации с подключенных к ним сенсоров и помещение соответствующих записей в хранилище); (3) базы данных с событиями безопасности (хранилища, содержащие записи о подозрительных событиях, которые в связке с http-сервером, например Apache, nginx, предоставляют доступ к их содержимому со стороны клиентской машины); (4) интерфейс просмотра событий безопасности (Prewikka-interface; frontend-интерфейс, используемый в системе для просмотра сработавших предупреждений и представленный Web-браузером с доступом через порт 8000). На рисунке А.5 показан пример взаимодействия компонентов Prelude [208]. Изображены 2 подсети, кото-

рые включают три и два сенсора, каждый из которых находится под управлением соответствующего Prelude-менеджера с отдельной консолью доступа. Центральный Prelude-менеджер выполняет функции по объединению результатов работы подчиненных ему Prelude-менеджеров и локальных для него сенсоров.

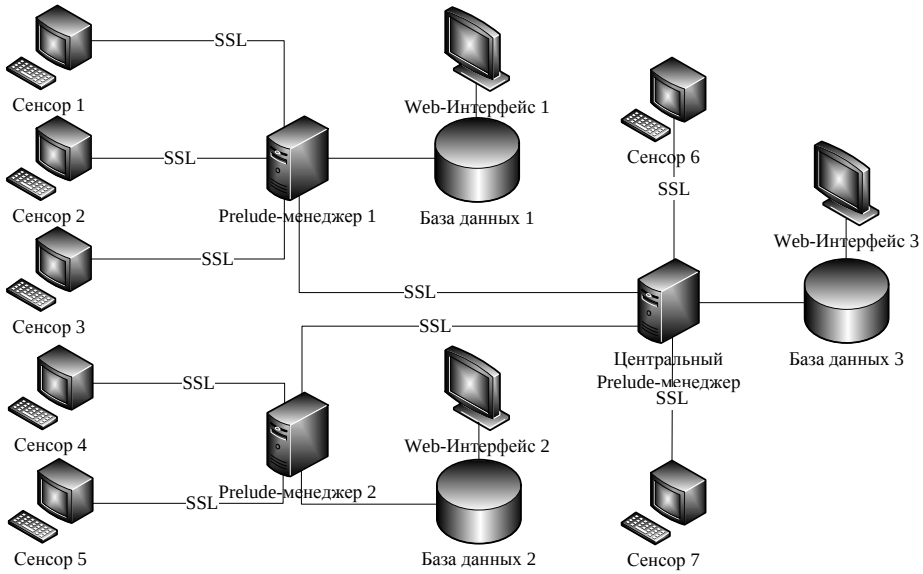


Рисунок А.5 — Схема взаимодействия компонентов Prelude

Разработчики системы не обошли стороной вопрос о необходимости создания безопасного канала для взаимодействия между компонентами. Так, при наличии на устанавливаемых хостах библиотеки OpenSSL, передача данных между хостами будет осуществляться через шифрованные соединения. При регистрации нового сенсора сперва генерируется сертификат, который потребуется для проверки подлинности источника данных во время его последующих подключений. Далее обмен данными происходит уже посредством туннелирования сообщений поверх TLS/SSL. В системе имеется в наличии набор API для возможности быстрой реализации сенсоров с необходимой функциональностью. За основу сенсоров берется готовая библиотека libprelude, среди функций которой можно назвать взаимодействие с Prelude-менеджером, создание шифрованного канала, а также формирование и отправка IDMEF-сообщений в рамках этого канала. Для ускорения передачи сообщений обмен данными осуществляется в бинарном виде [210] в соответствии с внутренним представлением формата IDMEF, описанным в libprelude как структура на языке C. В данный момент существуют байндинги к таким АЯВУ, как C, C++, Perl, Python, Ruby, Lua.

А.7 Сравнение характеристик СОА

В таблице 13 представлены сравнительные характеристики и их значения для каждой из представленных выше СОА. Сравнение СОА проводилось по следующим показателям: уровень мониторинга, тип реагирования, адаптивная способность, архитектура, метод обнаружения, поддерживаемые для развертывания платформы, принцип построения, способ расширения функциональности.

Таблица 13 — Сравнительные характеристики СОА

Характеристика	Snort	Suricata	Bro	OSSEC	Prelude
Уровень мониторинга	Сетевой	Сетевой	Сетевой	Хостовый	Сетевой и хостовый
Тип реагирования	Активный	Активный	Пассивный	Активный	Активный
Способность к адаптации	Присутствует в виде дополнительно подключаемых модулей	Присутствует в виде дополнительно подключаемых модулей	Присутствует как статистический анализ	Отсутствует	Отсутствует
Архитектура	Централизованная	Централизованная	Централизованная	Распределенная	Распределенная
Поддерживаемые платформы	Windows, *nix	Windows, *nix	*nix	Windows, *nix	Windows, *nix
Принцип построения	Монолитный	Монолитный	Монолитный	Компонентный	Компонентный
Способ расширения функциональности	Динамически подгружаемые декодеры	Динамически подгружаемые so/dll-библиотеки, Lua-скрипты	Bro-сценарии, so-библиотеки	XML-правила	Подключаемые сенсоры

Почти все описанные СОА являются активными. Исключением из этого списка является лишь Bro, которая не имеет встроенных средств для предотвращения атак, однако при помощи идущего в поставке с ней модуля exec.bro

можно настроить принудительный сброс подозрительного соединения или блокировку трафика на уровне ядра при помощи правил iptables. Это нисколько не обедняет данную платформу, поскольку согласно источнику [186] Bro разрабатывалась в первую очередь именно для изучения характеристик сетевого трафика, а не для обнаружения в нем атак. С этой целью в ней были реализованы мощные анализаторы протоколов, позволившие идентифицировать тип протокола даже на нестандартных портах благодаря механизму DPD (Data Packet Detection, Dynamic Protocol Detection), который аналогичен применяемому в Suricata. Кроме того, при разработке Bro сказалось ее прошлое: она развивалась в академическом окружении, поэтому характеризуется свойством адаптивности за счет наличия в ее основе статистических модулей обнаружения сетевых аномалий: попыток сканирования портов (scan.bro), проведения DoS-атак (synflood.bro) и т.д.

Все из представленных СОА являются кроссплатформенными, за исключением Bro, которая предназначена для запуска только в Unix-подобных ОС. Также отметим, что для платформы Windows система OSSEC не имеет режима локальной установки, т.е. для этой ОС необходимо устанавливать в отдельности и сервер, и агенты. Для ОС Linux, Mac OS и BSD такого ограничения нет [116]. И в то же время именно OSSEC является единственной из представленных СОА, способной обнаруживать вредоносные процессы как последствия успешно реализованных атак.

Для систем Snort и Suricata свойство адаптивности зачастую реализуется через дополнительные модули. Встречаются как отечественные [15], так и зарубежные [140, 146] исследования, в которых авторы встраивают в Snort такие интеллектуальные декодеры, к примеру, на основе нейронных сетей [15], метода опорных векторов [146] и деревьев решений [140].

По принципу построения можно классифицировать данные системы как монолитные и компонентные. В первом случае система представлена как единый бинарный файл, запуск которого средствами ОС, как правило, порождает не более одного процесса. К их числу из рассматриваемых СОА относятся

Snort, Suricata и Bro. Компонентный подход подразумевает разбиение системы на несколько функциональных блоков, каждый из которых запускается в отдельном пространстве пула адресов памяти, выполняет конкретную задачу и общается с остальными на основе механизмов межпроцессного взаимодействия. К ряду таких систем принадлежат OSSEC и Prelude.

Как уже было отмечено ранее, основными элементами для создания дополнительных функций в работе систем Snort и Suricata являются декодеры и препроцессоры, основа функционирования которых заключается в вызове определенных функций из динамически подгружаемых библиотек (so/dll). Наиболее продвинутым способом создания новых модулей, расширяющих функциональные возможности системы, обладает Bro. Вся рутинная работа по написанию шаблонных файлов будущего приложения сведена до минимума. Вместе с исходными кодами данной системы поставляется bash-скрипт init-plugin, предназначенный для автоматической генерации начального скелета будущего модуля, правил его сборки и установки. После компиляции модуль представляет собой готовую so-библиотеку, прототипы функций из которой обернуты в соответствующий bro-скрипт. Последующая загрузка, инициализация указателей на функции и деаллокация модулей системой сводятся к вызову функций dlopen, dlsym и dlclose соответственно.

Среди рассмотренных СОА наиболее полными характеристиками обладает Prelude. Будучи спроектированной изначально распределенной системой, она поддерживает гибридный мониторинг контролируемых узлов, осуществляя анализ как на уровне сети, так и на уровне хоста. Кроме того, эта система является масштабируемой, что позволяет ей использовать множество разнородных источников для сбора и обработки данных. Модульный принцип установки этой системы также позволяет добиться более гибкой настройки каждого из ее компонентов в отдельности. Возможность подключения каждой из представленных СОА в качестве сенсоров для Prelude (для Bro необходимо реализовывать нового клиента вручную, т.к. для нее нет готового сенсора) ставит ее на ранг выше остальных СОА.

А.8 Программные пути улучшения функционирования СОА

В данном разделе кратко перечислены хорошо известные способы, позволяющие повысить определенные качественные и количественные показатели функционирования СОА. Отметим, что применение названных способов не затрагивает существенной реорганизации архитектуры СОА или перестройки ее отдельных компонентов.

Программные пути улучшения функционирования СОА:

- использование технологии PF_RING для ускорения перехвата пакетов и „проброса“ их в пространство пользователя на основе операций zero copy, которую в данный момент поддерживают Snort, Suricata и Bro как функцию в драйвере сетевой карты Intel [164, 165, 175, 197];
- использование технологии XDP (eXpress Data Path);
- применение обработки агрегированных потоков, поступающих с нескольких netflow-сенсоров на коллектор с установленной СОА по протоколу IPFIX, для возможности быстрого встраивания распределенной архитектуры (пакет flow-tools);
- применение сетевых фильтров и экранов ipchains, pf, ipfw, iptables для блокировки подозрительных соединений и отсеивания ненужных пакетов;
- балансировка сетевой нагрузки с целью распределения обрабатываемого потока между несколькими параллельно запущенными на одной машине копиями СОА, а именно создание нескольких фиктивных (dummy) интерфейсов для перенаправления трафика на один из них, к примеру по правилу iface_num = src_ip % n_ifaces, каждый из которых прослушивается конкретной СОА;
- использование параллельных модификаций алгоритмов группового поиска шаблонной подстроки (к примеру, алгоритма Ахо–Корасик).

А.9 Обнаружение атак со скрытием и со вставкой

В данном разделе представляются результаты тестирования рассматриваемых открытых СОА, а именно их способности к обнаружению атак со скрытием и со вставкой, рассматриваются сценарии соответствующих атак. Кроме того, описывается архитектура, программная реализация, алгоритм функционирования разработанного программного средства, предназначенного для генерации таких атак, а также предлагается ряд алгоритмов, которые обеспечивают сборку IP-потоков и TCP-сессий из сырых пакетов, позволяют учитывать свойственные сетевому стеку ОС Linux особенности при обмене данными в сетях с использованием семейства протоколов TCP/IP и избежать пропуска некоторых типов атак со скрытием и со вставкой. Обсуждается соответствие экспериментальных результатов, полученных при оценке ОС (Linux и Windows) и сетевых СОА (Snort, Suricata и Bro), документам RFC.

Для проведения тестов был построен следующий программный макет, изображенный на рисунке А.6. В виртуальном окружении KVM (Kernel-based Virtual Machine) версии 1:2.1+dfsg-12+deb8u6 (deb-пакет qemu-kvm) были развернуты три машины: первый хост является атакующим (IP-адрес 10.0.1.100), второй (IP-адрес 10.0.1.101) и третий (IP-адрес 10.0.1.102) предоставляют доступ к некоторым запущенным на них сетевым службам и включают в свой состав установленную на каждый из этих хостов сетевую СОА. В качестве сервисов использовались HTTP (Web) и FTP, доступ к которым осуществлялся при помощи соответственно серверных приложений apache2 2.4.20 и vsftpd 3.0.2-17 для Linux 3.16.0-4, nginx 1.10.0 и MS FTP Service для Windows 7 со стороны защищаемых машин и тонких клиентов — разработанного и упрощенного интерактивного аналога wget/curl и программы ftp, запущенной в пассивном режиме (такой режим необходим для того, чтобы именно тонкий клиент выбирал порт, предназначенный для обмена данными), со стороны атакующей машины.

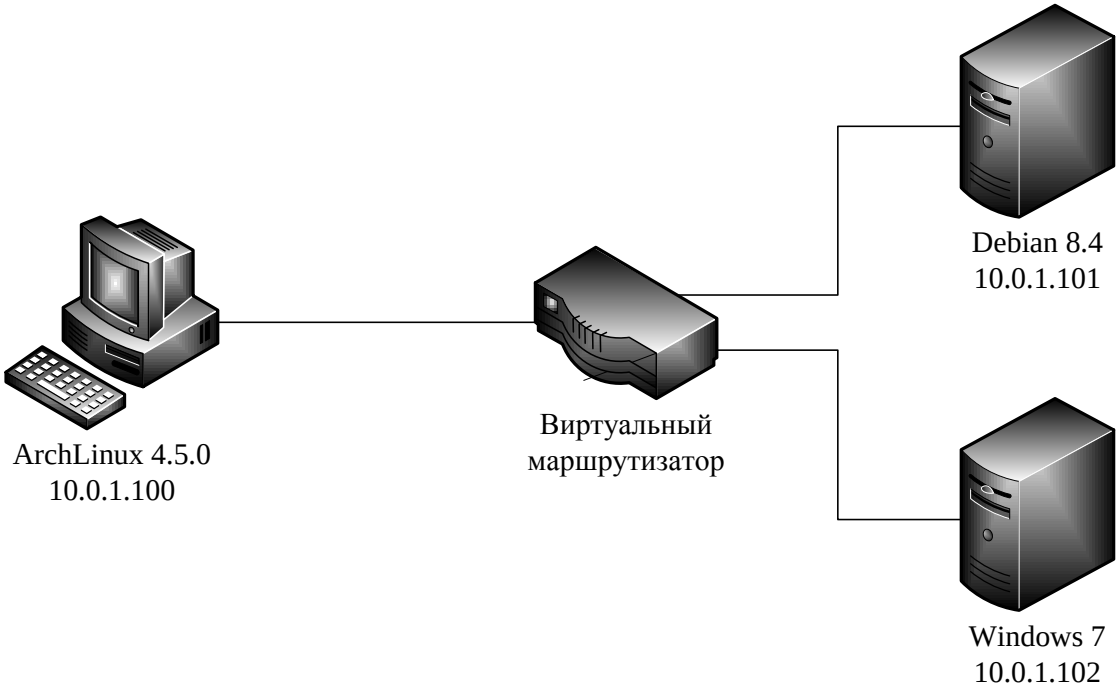


Рисунок А.6 — Программный стенд для тестирования сетевых СОА

Для тестирования сетевых СОА было разработанное специальное программное средство, предназначенное для прозрачного проксирования асинхронно поступающих клиентских запросов внутри TCP-соединений с поддержкой генерации тестовых последовательностей низкоуровневых сетевых атак и позволяющее осуществлять автоматизированное выполнение ряда сетевых атак на защищаемый хост с целью обхода одной из запущенных на нем СОА. Данный инструмент называется асинхронным прозрачным прокси-сервером TCP-сессий, общая архитектура которого представлена на рисунке А.7. В его состав входят три компонента. Первый компонент pkt_reader представляет собой сниффер (аналог tcpdump) с расширенным набором функций, который помимо чтения и разбора сетевых пакетов из интерфейса и pcap-файла способен „проигрывать“ сетевой трафик в интерфейс с заданной скоростью отправки, выполнять балансировку входящего трафика, изменять IP-адреса источника и назначения с пересчетом контрольных сумм, а также осуществлять фильтрацию определенных пакетов согласно заданному BPF-выражению. Особенностью этого компонента является возможность записи значений полей внутри захваченных пакетов в FIFO-канал в строковом виде. Второй компонент pkt_sender предназначен для заполнения полей IP-пакета с любым вложенным в него типом протокола (TCP,

UDP, ICMP или формируемый вручную пользователем) и дальнейшей отправки этого пакета по сырому сокету (среди открытых инструментов с подобным набором функционала можно назвать sendip и hping3). Этот компонент может быть запущен как в обычном режиме, так и в демон-режиме. В первом случае сведения о пакете формируются из аргументов командной строки, во втором случае управляющие команды считываются из именованного UNIX-канала. Для записи потока бинарных данных или непечатаемых символов в FIFO-канал применяется преобразование, схожее с percent-encoding. Роль третьего компонента pkt_processor заключается в выполнении подмены некоторых пакетов и обеспечении взаимодействия между первыми двумя компонентами.

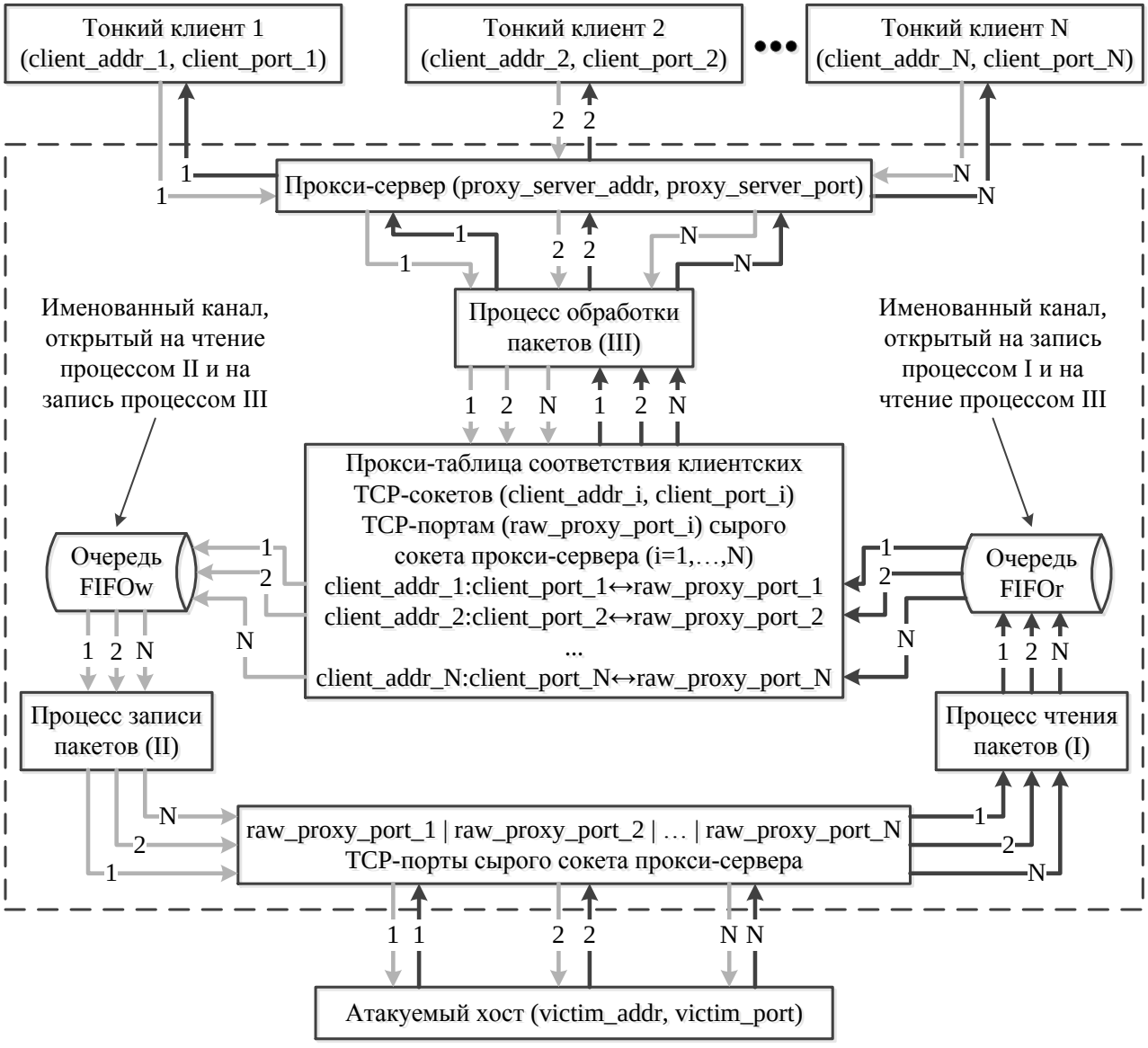


Рисунок А.7 — Архитектура асинхронного прозрачного прокси-сервера TCP-сессий

На рисунке А.7 изображены N тонких клиентов, которые подключаются к прокси-серверу через стандартные TCP-сокеты. Сам прослушивающий их прокси-сервер может быть запущен как локально, так и на отдельно выделенной машине. Выбор клиентских портов может осуществляться как встроенными средствами ОС при помощи вызова функции connect с переданными в нее аргументами дескриптора клиентского сокета и адреса прокси-сервера вместе с портом, так и вручную при помощи последовательности вызовов функций bind и connect. На стороне прокси-сервера запускается три процесса, соответствующих вышепредставленным компонентам: pkt_reader, pkt_sender и pkt_processor. Компонент pkt_processor обрабатывает входящие от клиентских приложений пакеты и осуществляет поиск в каждом из таких пакетов заданной сигнатурной последовательности. В случае ее вхождения в контент захваченного пакета последний подвергается определенным преобразованиям, которые зависят от типа выбранного злоумышленником сценария атаки (скрытая IP-фрагментация, отправка пакетов с задержками, установка противоречивых флагов в заголовке TCP-пакета, изменение порядка следования пакетов и т.д.). При подключении нового i-ого клиента (client_addr_i, client_port_i) на порту proxy_server_port прокси-сервера рассматриваемый компонент создает i-ую запись в прокси-таблице. В такой таблице представлены данные обо всех зарегистрированных на порту proxy_server_port входящих соединениях. Одновременно с добавлением этой записи в прокси-таблицу осуществляется поиск и резервирование незанятого порта raw_proxy_port_i, и назначается его соответствие i-ому клиентскому сокету. Именно через порт raw_proxy_port_i выполняется эмулированное при помощи сырого сокета сетевое взаимодействие между прокси-сервером и атакуемым хостом (victim_addr, victim_port). Как только приходят пакеты от i-ого клиента (или от атакуемого хоста на порт raw_proxy_port_i прокси-сервера), выполняется автоматическое перенаправление этих пакетов согласно i-ой записи в прокси-таблице. Тем самым для клиента обмен данными с атакуемым хостом происходит незаметно („прозрачно“) через некоторого промежуточного посредника (прокси-сервера), предоставляющего услуги по трансформации пакетов.

В отличие от популярных инструментов fragrouter и fragroute, которые также предназначены для выполнения атак со скрытием и со вставкой, в разработанном программном средстве используется клиент-серверная архитектура, и задействуется существенно больший перечень разнообразных сценариев атак.

На рисунке А.8 представлен алгоритм функционирования асинхронного прозрачного прокси-сервера TCP-сессий. Первый компонент (pkt_reader) передает в очередь FIFOr пакеты, поступающие от атакуемого сервера, в постобработанном строковом представлении. Второй компонент (pkt_sender) создает сетевой канал, в котором данные передаются минуя ОС на уровне сырых сокетов. Также необходимо отметить, что посредством задания правил стандартного встроенного в Linux файерволла на машине с установленным прокси-сервером была запрещена отправка исходящих RST-пакетов с той целью, чтобы сетевой стек ОС игнорировал входящие пакеты на закрытый порт сырого сокета (он является закрытым, поскольку он не зарегистрирован в ОС как используемый), и т.с. у удаленного сервера создавалась видимость, что он общается с реальным приложением, хотя на самом деле осуществляется некоторое преобразование пакетов, переадресуемых от тонких клиентов. Правило, блокирующее отправку таких пакетов на уровне ядра с исходящим портом raw_proxy_port_i, задается при помощи следующей команды: iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP --sport raw_proxy_port_i. Команда, которая отменяет введенные предыдущим правилом изменения, — iptables -D OUTPUT -p tcp --tcp-flags RST RST -j DROP --sport raw_proxy_port_i. Третий компонент (pkt_processor) запускает TCP-сокет с прослушиванием входящих соединений на порту 9000 и „прозрачно“ перенаправляет запросы от подключаемых к нему тонких клиентов на удаленный атакуемый сервер. С этой целью он записывает данные в строковое представление пакетов в очередь FIFOw второго компонента, который подхватывает эту запись, формирует реальный пакет и отправляет в интерфейс на сторону удаленного сервера. Перед непосредственной отправкой на сервер некоторых пакетов происходит их изменение по одному из правил, задающих описанные ниже сценарии атак.

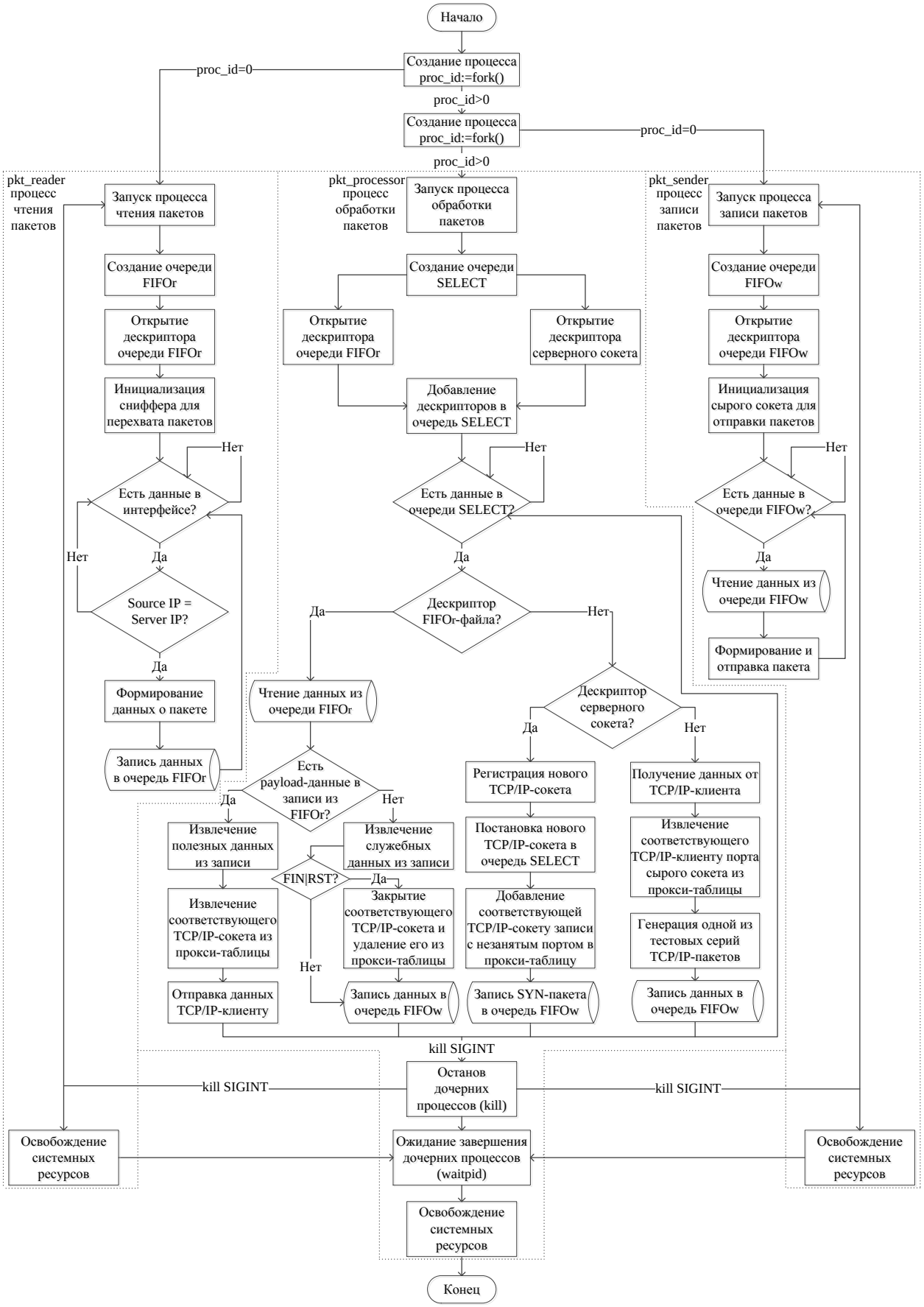


Рисунок А.8 — Алгоритм асинхронного прозрачного проксирования сетевых соединений

При регистрации нового входящего от тонкого клиента соединения на входе асинхронного прозрачного прокси-сервера TCP-сессий не создается нового потока, вместо этого дескриптор соответствующего клиентского сокета ставится в очередь SELECT (EPOLL) для мониторинга его состояния и проверки готовности данных к прочтению из этого дескриптора. Поскольку все функционирование прокси-сервера заключено в один цикл while ($select_queue→can_read()), то экономятся вычислительные ресурсы посредством снижения расходов, связанных с созданием новой нити для обработки клиентских соединений.

Особенности разработанного программного средства для проведения атак со скрытием и со вставкой следующие:

- частичная эмуляция TCP/IP-взаимодействия на уровне сырых сокетов:
  - автоматическое вычисление контрольных сумм на IP- и TCP-уровнях;
  - автоматическое вычисление значений следующих позиционных номеров и номеров подтверждения;
  - корректное установление и завершение TCP-сеанса;
  - автоматическое вычисление IP-идентификаторов;
  - автоматическое вычисление смещений для IP-фрагментов;
- прозрачное проксирование TCP/IP-соединений;
- поддержка множественности подключений от различных клиентов;
- асинхронная обработка поступающих от тонких клиентов запросов;
- однопоточный режим работы (т.с. нет необходимости „плодить“ множество процессов или потоков на каждое входящее от клиента соединение);
- модульность построения (все низкоуровневые части для работы с сетью написаны на C, компонент их взаимодействия реализован через скрипт на Perl);
- генерация тестовых последовательностей низкоуровневых сетевых атак, в основном направленных на фрагментацию пакетов, установку всевозможных значений полей заголовков от самого низкого уровня до самого высокого.

В дальнейшем планируется добавить обработку и формирование IP- и TCP-опций.

Для проведения сетевых атак были подготовлены 25 сценариев, которые реализуют различные слабости или не жестко установленные спецификациями ограничения протоколов. Выбор именно таких сценариев атак обусловлен их несложной реализацией и наилучшим покрытием нескольких, а именно четырех, уровней модели OSI. Первый сценарий затрагивает канальный уровень, сценарии со второго по десятый реализуют не явно регламентированные особенности IP-протокола сетевого уровня, сценарии с номерами 11–24 описывают некоторые возможные бреши в построении сетевого стека СОА на транспортном уровне, двадцать пятый сценарий описывает возможную атаку на прикладном уровне. Всего было проведено 240 различных экспериментов. Ниже представлены краткие сведения о рассматриваемых сценариях атак.

1. Подмена аппаратного адреса отправителя для одного из пакетов установленного TCP/IP-сеанса („спуфинг MAC-адреса отправителя“).
2. Построение и отправка одного из пакетов TCP/IP-сеанса с неправильно вычисленной контрольной суммой IP-заголовка.
3. Отправка одного из пакетов TCP/IP-сеанса в нескольких IP-фрагментах в порядке убывания значений IP-поля смещения.
4. Разбиение одного из пакетов TCP/IP-сеанса на уровне протокола IP на несколько пакетов с частично перекрывающимися при их склейке данными (экспериментально было установлено, что для Linux в случае частично перекрывающихся данных большее предпочтение отдается пакетам, поступившим раньше, а точнее имеющим меньшее значение поля IP-offset внутри исходной дейтаграммы; Windows вовсе не обрабатывает частично перекрывающиеся пакеты).
5. Разбиение одного из пакетов TCP/IP-сеанса на уровне протокола IP на три фрагмента: первый фрагмент содержит неправильную контрольную сумму IP-заголовка и произвольно сгенерированные данные, второй фрагмент содержит последнюю половину сигнатурных данных, тре-

тий фрагмент полностью перекрывает первый фрагмент, но на этот раз содержит правильную контрольную сумму IP-заголовка и первую половину сигнатурных данных (стек ОС игнорирует первый фрагмент, заменяет его третьим и склеивает со вторым).

6. Разбиение одного из пакетов TCP/IP-сеанса на уровне протокола IP на три фрагмента: первый фрагмент содержит произвольно сгенерированные данные, второй фрагмент имеет такие же параметры, как и первый фрагмент, но с нужной первой половиной сигнатурных данных (и т.с. полностью его перекрывает), третий фрагмент содержит вторую половину сигнатурных данных (экспериментально было проверено, что для Linux в случае полностью перекрывающихся IP-фрагментов больший приоритет устанавливается пакетам, поступившим позже; для Windows — наоборот).
7. Отправка одного из IP-фрагментов с заданной временной задержкой (в экспериментах использовалось значение таймаута, не меньшее 30 сек. и 59 сек., используемое соответственно в Linux и Windows по умолчанию перед отправкой ICMP-пакета с типом 11 «Time-to-live exceeded» и кодом 1 «Fragment reassembly time exceeded»).
8. Отправка IP-дейтаграммы с пустыми флагами.
9. Отправка IP-дейтаграммы со значением поля длины IP-заголовка, меньшим 20 байт.
10. Отправка IP-дейтаграммы с максимальным значением поля длины IP-пакета, равным 65535 байт.
11. Построение и отправка одного из пакетов TCP/IP-сеанса с неправильно вычисленной TCP-контрольной суммой.
12. Отправка нескольких TCP-пакетов в обратном порядке следования позиционных номеров.
13. Отправка TCP-пакетов с частичным наслоением на ранее отправленные данные внутри TCP-пакетов (экспериментально было установлено, что

для Linux и Windows больший приоритет устанавливается пакетам с наименьшим позиционным номером).

14. Разбиение одного из пакетов TCP/IP-сеанса на уровне протокола TCP на три сегмента: первый сегмент содержит неправильную контрольную сумму TCP-псевдозаголовка и произвольно сгенерированные данные, второй сегмент содержит последнюю половину сигнатурных данных, третий сегмент полностью перекрывает первый сегмент, но на этот раз содержит правильную контрольную сумму TCP-псевдозаголовка и первую половину сигнатурных данных (стек ОС игнорирует первый сегмент, заменяет его третьим и склеивает со вторым).
15. Разбиение одного из пакетов TCP/IP-сеанса на уровне протокола TCP на три сегмента: первый сегмент содержит произвольно сгенерированные данные и пустые флаги (прием пакета без установленного ACK-флага в рамках уже установленной TCP-сессии игнорируется ОС), второй сегмент содержит вторую половину сигнатурных данных, третий сегмент имеет такие же параметры, как и первый сегмент, но с нужной первой половиной сигнатурных данных (и т.с. полностью его перекрывает) и с установленным флагом ACK (ОС успешно подтверждает прием третьего сегмента и склеивает его со вторым).
16. Отправка одного из TCP-сегментов с заданной временной задержкой (в экспериментах использовалось значение таймаута, не меньшее 20 сек., используемое в apache2 для Linux по умолчанию перед отправкой FIN-пакета для терминирования соединения; в случае Windows это же значение таймаута не разрывало соединения с сервером nginx).
17. Отправка полезных данных в первом TCP-пакете сессии с установленным SYN-флагом (опция сокета MSG_FASTOPEN, RFC 7413 [79]).
18. Отправка полезных данных в третьем пакете в рамках „тройного рукопожатия“ TCP-машины состояний с установленным ACK-флагом.
19. Отправка полезных данных в FIN-пакете TCP-сессии.
20. Отправка полезных данных в RST-пакете TCP-сессии.

21. Отправка полезных данных в TCP-пакетах со сброшенными флагами.
22. Отправка полезных данных в TCP-пакете с установленными URG|FIN|PSH|ACK-флагами («XMAS-tree», „рождественская елка“).
23. Отправка TCP-пакета со значением поля длины TCP-заголовка, меньшим 20 байт.
24. Отправка данных в TCP-пакете с позиционным номером, меньшим начального позиционного номера (ISN), полученного от инициатора сессии („вываливание за пределы TCP-сессии“).
25. Вставка управляющих символьных последовательностей перед командами загрузки в текстовых прикладных протоколах HTTP и FTP (в экспериментах использовалась последовательность шестнадцатеричных символьных кодов 0x0D 0x0A 0x0D 0x0A ("\r\n\r\n"), означающая завершение команды, в качестве префикса перед GET и RETR).

Атаки, связанные с манипуляцией значения IP-поля TTL, не рассматриваются, т.к. СОА и защищаемая ей ОС находятся на одном хосте. За более подробным описанием других различных техник обмана сетевых СОА читатель может обратиться к классическому труду Ptacek и Newsham [180].

В качестве основы сигнатур использовалось стандартное правило Snort с идентификатором sid=1762 для обнаружения известной уязвимости Web-сервера, которая позволяет выполнить произвольный код на удаленной машине с правами суперпользователя посредством запуска CGI-скрипта. Для удобства проведения экспериментов было использовано простое правило, которое построено на базе вышеупомянутого правила и обнаруживает доступ к файлам secret.html и confidential_info.txt соответственно через сервисы HTTP и FTP. В первом случае осуществлялся поиск символьной последовательности "GET /secret.html", во втором случае искомой строкой являлась "RETR confidential_info.txt". Данные команды, генерируемые клиентом, предназначены для формирования запроса на загрузку файлов secret.html и confidential_info.txt, расположенных в корне соответственно Web-сервера apache2 (к примеру /var/www/html) и FTP-сервера vsftpd (к примеру, для пользователя anonymous название корневой директории

настраивается через параметр anon_root в файле /etc/vsftpd.conf). Понятно, что вместо таких простых команд могут быть применены более сложные атаки, как например атаки, которые реализуют уязвимости CVE-1999-0067 (удаленное выполнение команд при помощи служебных символов командной оболочки) или CVE-2015-7767 (переполнение буфера, позволяющее выполнить произвольный код или вызвать отказ в обслуживании). Также отметим, что не рассматриваются атаки, которые направлены на примитивный обход сигнатурного правила, к ряду которых относятся нарочная вставка дополнительных незначащих пробелов между командой и ее параметрами, изменение регистра в командах и их аргументах или замена стандартного пути до файла каким-либо другим эквивалентным абсолютным или относительным путем до этого же файла.

Соответствующие правила для Snort и Suricata представлены ниже.

```
# test_examples.rules
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 \
(msg:"WEB test attack"; \
 flow:to_server,established; \
 content:"GET /secret.html"; \
 fast_pattern; \
 sid:1000003; \
 rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 \
(msg:"FTP test attack"; \
 flow:to_server,established; \
 content:"RETR confidential_info.txt"; \
 fast_pattern; \
 sid:1000004; \
 rev:1;)
```

Соответствующие правила для Bro, частично сгенерированные с помощью Python-скрипта snort2bro из вышепредставленных Snort-сигнатур, даны ниже.

```
# test_examples.sig
signature 1000003-1 {
  ip-proto == tcp
  src-ip != 10.0.1.101 # local_nets
  dst-ip == 10.0.1.101 # local_nets
  dst-port == 80
  event "WEB test attack"
  tcp-state established,originator
  payload /.*GET \/secret\.html/
  }

signature 1000004-1 {
  ip-proto == tcp
  src-ip != 10.0.1.101 # local_nets
  dst-ip == 10.0.1.101 # local_nets
  dst-port == 21
  event "FTP test attack"
  tcp-state established,originator
  payload /.*RETR confidential_info\.txt/
  }

# test_examples.bro
@load base/frameworks/signatures
@load-sigs ./test_examples.sig
module TestExamples;
global log_file: file;

event bro_init()
    {
    log_file = open(fmt("test_examples_%s.log",
                        current_time()));
    }

event bro_done()
    {
    close(log_file);
    }

event signature_match(state: signature_state,
                      msg: string, data: string)
                      &priority=5
    {
    print log_file, fmt("%s alert %s", current_time(),
                        msg);
    flush_all();
    }
```

В таблице 14 представлены результаты тестирования СОА на их способность обнаружения представленных сценариев атак на примере защищаемого хоста под управлением Debian Jessie 8.4. Дистрибутив Linux, установленный на атакующем хосте, — ArchLinux 4.5.0. Прокси-сервер запускался локально на этом же хосте. Знаком «+» отмечены те сценарии, которые вызывают срабатывание правила СОА или приводят к получению ответного пакета от сервера. Знак «−» означает отсутствие срабатывания правила для СОА или отклика от сервера на входящий пакет. Знаком «∗» обозначены те сценарии, которые либо (1) на самом деле не являются атакующими действиями, но при этом СОА распознает их как атаку и оповещает об аномальности таких пакетов („эффект избыточной предосторожности“), либо (2) на самом деле являются атакующими действиями, но при этом СОА распознает их с большой временной задержкой, или запись об атаке появляется только после останова СОА, а именно после сброса буфера в дескриптор лог-файла („эффект запоздалой реакции“). В частности, для более удобного представления символ «⊖» означает пропуск атаки, а символ «⊞» — ложное срабатывание. Знаком «?» помечены нехарактерные для сервиса сценарии, которые являются невозможными в силу его специфики, к примеру для протокола FTP невозможна загрузка файла без выполнения предварительной процедуры авторизации на сервере, поэтому сценарии 17 и 18 для этой службы опущены. Результатом успешно проведенной и в то же время незамеченной атаки является появление на атакующей машине файла, загруженного с Web- или FTP-сервера, и отсутствие записи об этой атаке в журнале СОА. Для Debian каждая из рассматриваемых в данном разделе СОА (Snort, Suricata, Bro) была скомпилирована из исходных кодов и собрана в установочный deb-пакет при помощи инструмента checkinstall с ключом -D. Все три СОА запускались с настройками по умолчанию, в которые уже были включены опции обработки и сборки фрагментированных IP-пакетов. Дополнительно к этим системам были добавлены вышеописанные сигнатурные правила для обнаружения подстрок "GET /secret.html" и "RETR confidential_info.txt" в контенте HTTP- и FTP-запросов.

Таблица 14 — Результаты тестов СОА на способность обнаружения сценариев атак (Debian Jessie 8.4)

Сценарии атак	Snort 2.9.8.2		Suricata 3.0.1		Bro 2.4.1		Сетевой стек Linux 3.16.0-4	
	apache2	vsftpd	apache2	vsftpd	apache2	vsftpd	apache2	vsftpd
Сценарий 1	+	+	+	+	+	+	+	+
Сценарий 2	−	−	−	−	−	−	−	−
Сценарий 3	+	+	+	+	+	+	+	+
Сценарий 4	+	+	+	+	+	+	+	+
Сценарий 5	+	+	⊖	⊖	+	+	+	+
Сценарий 6	⊖	⊖	+	+	⊖	⊖	+	+
Сценарий 7	⊞	⊞	−	−	⊞	⊞	−	−
Сценарий 8	+	+	+	+	+	+	+	+
Сценарий 9	−	−	−	−	−	−	−	−
Сценарий 10	−	−	−	−	−	−	−	−
Сценарий 11	−	−	*	*	−	−	−	−
Сценарий 12	+	+	+	+	+	+	+	+
Сценарий 13	+	+	+	+	+	+	+	+
Сценарий 14	+	+	+	+	+	+	+	+
Сценарий 15	+	+	+	+	⊖	⊖	+	+
Сценарий 16	⊞	⊖	−	+	⊞	+	−	+
Сценарий 17	+	?	+	?	⊖	?	+	?
Сценарий 18	+	?	+	?	+	?	+	?
Сценарий 19	+	+	+	+	+	+	+	+
Сценарий 20	⊞	⊞	−	−	−	−	−	−
Сценарий 21	⊞	⊞	⊞	⊞	⊞	⊞	−	−
Сценарий 22	⊞	+	⊞	+	⊞	+	−	+
Сценарий 23	−	−	−	−	−	−	−	−
Сценарий 24	+	+	⊖	⊖	⊖	⊖	+	+
Сценарий 25	+	+	+	+	+	+	+	+

По результатам экспериментов, приведенных в таблице 14, можно сказать, что все три СОА успешно справляются с дефрагментацией и реассемблированием соответственно IP-фрагментов и TCP-сегментов (сценарии 3, 4, 12, 13). У двух СОА, Snort и Bro, есть два похожих недостатка при обработке пакетов, пришедших уже после выделенного для их склейки времени (сценарии 7 и 16). Suricata, в свою очередь, правильно отслеживает ICMP- и TCP-пакеты, сигнализирующие о закрытии канала передачи сетевого потока со стороны сервера (прием пакетов и подтверждение их получения допускается сервером).

В работе Snort было выявлено пять ложных срабатываний, которые соответствуют сценариям 7, 16, 20, 21 и 22. При помощи таких атак злоумышленник может специально вводить аналитиков безопасности в заблуждение, отвлекая их на напрасный поиск отсутствующей аномалии. Такие атаки являются частным случаем атак со вставкой: СОА реагирует на пакеты такого типа, но хостовая ОС игнорирует их обработку. Также для Snort было обнаружено, что она пропускает два сценария атак под номерами 6 и 16. В случае FTP для сценария 16 можно обойти Snort посредством задания значения таймаута 180 сек., которое используется в Snort как настройка по умолчанию для удаления просроченных TCP-сессий (FTP-сервер под управлением vsftpd допускает таймаут до 300 сек. по умолчанию). Экспериментально было проверено: пропуск системой Snort сценария атаки 6 обусловлен тем, что Snort использует характерный именно для Windows принцип склейки IP-фрагментов вне зависимости от того, под какой ОС она запущена (таблица 15), хотя внутри стеков этих ОС используется абсолютно противоположный друг другу механизм дефрагментации в случае конфликта, описанного в сценарии 6[7]. Отметим наперед, что для системы Suricata в случае этого же сценария наблюдается обратная картина: в ней реализован механизм склейки, свойственный именно Linux без привязки к хостовой ОС, на которой она запущена.

Систему Suricata отличает бóльшая информативность в выводе log-сообщений об обнаруженных сетевых аномалиях, в частности, она по умолчанию

[7]Для устранения этого недостатка необходимо явно указывать параметр policy для препроцессора frag3_engine.

уведомляет о пакетах с нарушенной контрольной суммой, с перекрывающимися частями IP/TCP-пакетов, с невалидной длиной заголовка. Было обнаружено два серьезных недостатка в работе данной СОА — пропуски сценариев атак под номерами 5 и 24, а также несколько ложных срабатываний, соответствующих сценариям под номерами 11, 21, 22. Для сценария 11 сама СОА выдает предупреждение о неправильной контрольной сумме TCP-псевдозаголовка. Отсюда можно судить, что в систему специально заложена некоторая избыточная предосторожность для слежки за атаками подобного уклонения вследствие того, что другие хостовые ОС, возможно, все-таки обработают такие пакеты. Поэтому можно не принимать во внимание ложное срабатывание Suricata, вызванное сценарием 11, который приводит к такому несоответствию в обработке пакетов этой СОА и сетевым стеком Linux. Кстати, это же предположение подтверждается тем фактом, что Suricata успешно обнаруживает попытку уклонения, заданную сценарием 14.

В ходе проведения экспериментов было установлено, что Bro пропускает атаки, задаваемые четырьмя сценариями 6, 15, 17, 24. Пожалуй, наиболее существенное слабое место в реализации сетевого анализатора Bro заключается в том, что он не обрабатывает данные, передаваемые в первом пакете TCP-сессии (сценарий 17). Даже после удаления ключевого слова established в соответствующей сигнатуре система по-прежнему игнорировала такие пакеты. Отметим, что для системы Snort обнаружение такого уклонения решилось отключением именно этой опции. Поэтому здесь возможно осуществление атаки со скрытием: СОА не видит аномального пакета, а целевой хост, в свою очередь, его воспринимает. Тем самым создается потайной ход, позволяющий злоумышленнику незаметно обойти СОА и без наказуемых для него последствий внедрить какой-либо вредоносный код на атакуемый сервер. Также для Bro было обнаружено, что она ошибочно считает за атаки четыре сценария под номерами 7, 16, 21, 22.

В таблице 15 представлены результаты тестирования СОА на их способность обнаружения представленных сценариев атак на примере защищаемого хоста под управлением Windows 7. Обозначения, применяемые в этой таблице, идентичны ранее введенным.

Таблица 15 — Результаты тестов СОА на способность обнаружения сценариев атак (Windows 7)

Сценарии атак	Snort 2.9.8.2		Suricata 3.0.1		Сетевой стек Windows 7	
	nginx	MS FTP Service	nginx	MS FTP Service	nginx	MS FTP Service
Сценарий 1	+	+	+	+	+	+
Сценарий 2	−	−	*	*	−	−
Сценарий 3	+	+	+	+	+	+
Сценарий 4	⊞	⊞	*	*	−	−
Сценарий 5	+	+	⊖	⊖	+	+
Сценарий 6	+	+	⊖	⊖	+	+
Сценарий 7	⊞	⊞	−	−	−	−
Сценарий 8	+	+	+	+	+	+
Сценарий 9	−	−	−	−	−	−
Сценарий 10	−	−	−	−	−	−
Сценарий 11	−	−	*	*	−	−
Сценарий 12	⊖	*	⊖	⊖	+	+
Сценарий 13	+	+	⊖	⊖	+	+
Сценарий 14	⊖	*	⊖	⊖	+	+
Сценарий 15	⊖	*	⊖	⊖	+	+
Сценарий 16	⊖	*	⊖	⊖	+	+
Сценарий 17	+	?	+	?	+	?
Сценарий 18	+	?	+	?	+	?
Сценарий 19	+	+	+	+	+	+
Сценарий 20	⊞	⊞	*	*	−	−
Сценарий 21	*	*	⊞	⊞	−	−
Сценарий 22	+	+	+	+	+	+
Сценарий 23	−	−	−	−	−	−
Сценарий 24	+	+	+	+	+	+
Сценарий 25	+	+	+	+	+	+

Установка двух СОА, Snort и Suricata, в среде Windows 7 выполнялась с использованием уже готовых exe-файлов. Для этих систем использовались, как и ранее, настройки по умолчанию, расширенные двумя новыми сигнатурными правилами с идентификаторами sid=1000003 и sid=1000004. Для защищаемого хоста Windows 7 в ходе экспериментов были получены следующие результаты. Snort пропускает четыре сценария атак под номерами 12, 14, 15, 16 и имеет три ложных срабатывания, вызванных сценариями 4, 7 и 20. Suricata, в свою очередь, пропускает семь сценариев атак под номерами 5, 6, 12, 13, 14, 15, 16 и одно ложное срабатывание, вызванное сценарием 21, относит к атакующим действиям. Сценарий 17 в ОС Windows 7 проверялся в режиме offline: сетевой трафик, содержащий TCP-соединение с установленным SYN-флагом в первом пакете и непустым контентом, был сгенерирован в среде ОС Debian Jessie 8.4, сохранен в формате pcap-файла, после чего этот файл был прочитан СОА Snort и Suricata, запущенными в Windows 7. В Linux для поддержки возможности обработки подобных пакетов необходимо выполнить следующие команды, запущенные от имени суперпользователя:

```
echo 1 > /proc/sys/net/ipv4/tcp_fastopen
sysctl -w net.ipv4.tcp_fastopen=3
systemctl restart systemd-sysctl
/etc/init.d/networking restart
```

Для устранения вышеописанных недостатков, обнаруженных в функционировании сетевых СОА (Snort, Suricata, Bro), был разработан событийно-ориентированный анализатор сетевого трафика, описание которого приведено в разделе 3.2. В его функционирование были добавлены алгоритмы IP-дефрагментации и TCP-реассемблирования, свойственные ОС семейства Linux и тем самым учитывающие возможные способы обхода СОА Snort, Suricata и Bro; блок-схемы этих алгоритмов приведены на рисунках А.9 и А.10. Кроме того, данный анализатор предоставляет удобный API для выявления попыток проведения некоторых атак со скрытием и со вставкой.

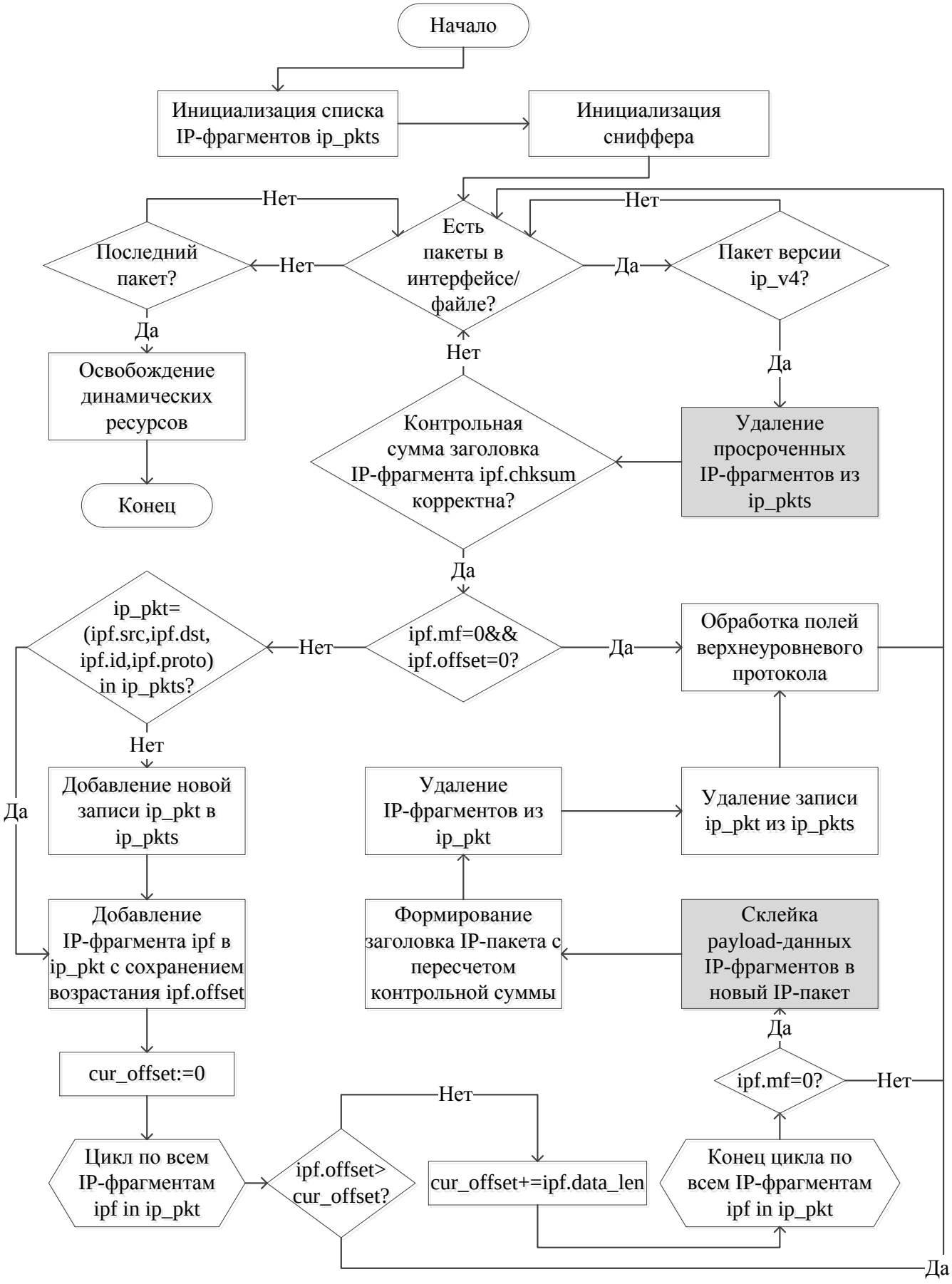


Рисунок А.9 — Дефрагментация IP-пакетов

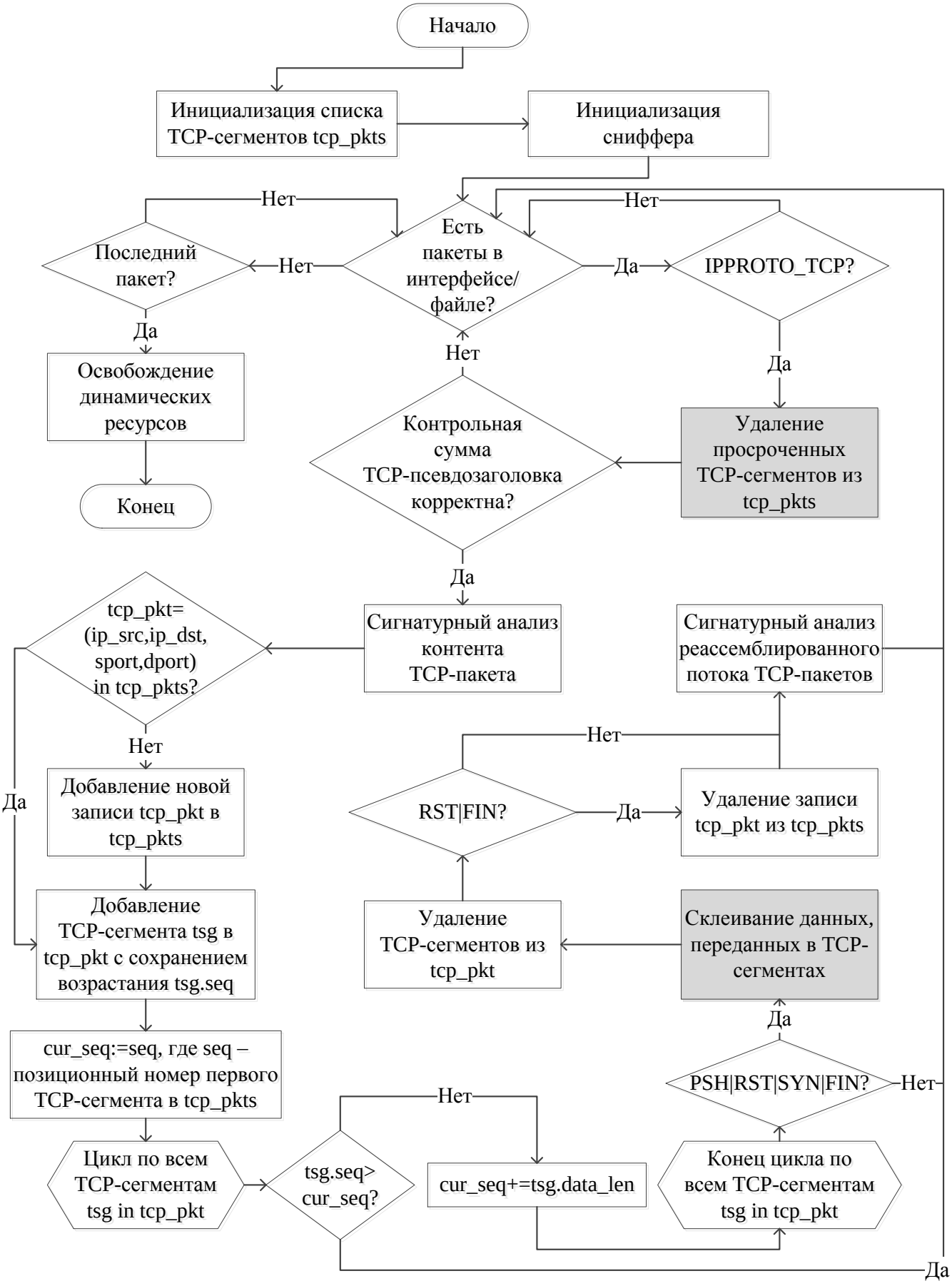


Рисунок А.10 — Реассемблирование данных, передаваемых в TCP-сегментах

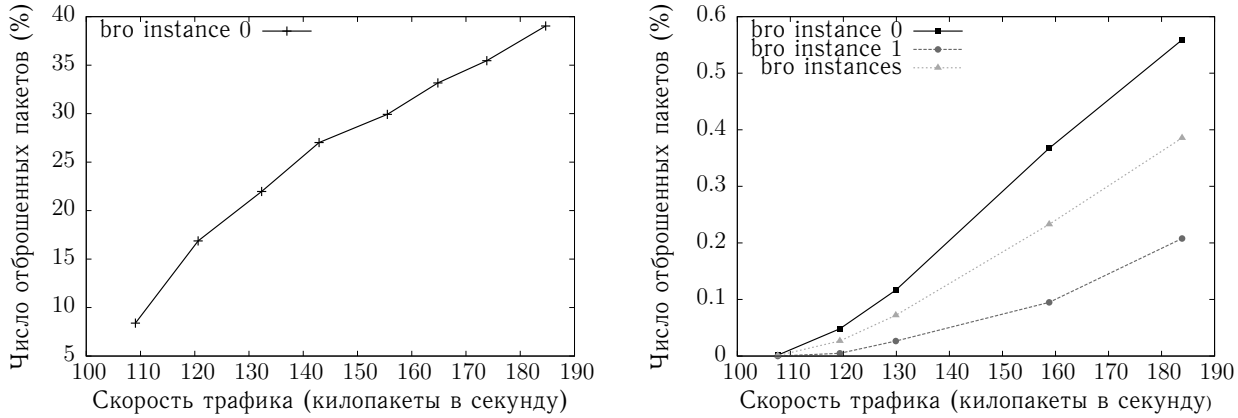
Рассмотрим наиболее тонкие особенности, присущие алгоритму IP-дефрагментации. Такие блоки выделены на рисунке А.9 темным цветом: „склейка payload-данных IP-фрагментов в новый IP-пакет“ и „удаление просроченных фрагментов из ip_pkts“. Некоторые из этих особенностей в первую очередь вызваны тем, что в соответствующих документах (RFC 791 [177], RFC 1858 [213], RFC 3128 [156]) нет строгих правил, точно задающих процесс унифицированной обработки таких конфликтующих пакетов, что приводит к некоторым различиям в программной реализации сетевых стеков на различных ОС. В частности, в RFC 791 [177] говорится о том, что в случае идентичных фрагментов или фрагментов, которые наслаиваются друг на друга, начиная с левой границы (т.е. имеют одинаковое значение IP-поля смещения), больший приоритет должен отдаваться пакетам поступившим позже, что в соответствии со сценарием 6 согласуется с поведением стека Linux. Однако, как было показано экспериментально, стек Windows противоречит этому требованию и действует прямо противоположно. В то же время для сценария 4 в RFC нет явных указаний для склейки пакетов, которые частично перекрываются, но при этом имеют разное значение IP-поля смещения. Поэтому здесь возникает некоторый произвол при выборе того, какому именно пакету устанавливать больший приоритет. Для корректной реализации процедуры удаления просроченных IP-фрагментов СОА должна либо отслеживать соответствующие ICMP-пакеты, которые сигнализируют об истечении таймаута, выделенного на сборку IP-пакета, либо иметь идентичное стеку ОС значение таймаута, к примеру, полученное из файла /proc/sys/net/ipv4/ipfrag_time. При вызове этой процедуры используется оригинальный прием, заключающийся в проверке просроченности фрагментов в порядке добавления их указателей в ip_pkts. TCP-реассемблирование применяется только к пакетам, прошедшим всю стадию полной дефрагментации на уровне IP. К примеру, сборку TCP-соединений можно начинать во время выполнения блока, названного на рисунке А.9 как „обработка полей верхнеуровневого протокола“. Для вычисления контрольной суммы автором настоящего диссертационного исследования разработана универсальная процедура, которая подходит для IP-, TCP-, UDP- и ICMP-пакетов:

```
#pragma pack(push)
#pragma pack(1)
typedef struct {
  uint32_t saddr;
  uint32_t daddr;
  uint8_t reserved; // 8 bits of zeroes
  uint8_t proto;
  uint16_t len; // length of header with data
  const uint8_t *data;
} pseudo_hdr;
#pragma pack(pop)
void fill_pseudo_hdr(pseudo_hdr *pseudo_header, uint32_t saddr,
                     uint32_t daddr, uint8_t proto,
                     const uint8_t *pkt, uint16_t pkt_len) {
  memset(pseudo_header, 0, sizeof(pseudo_hdr));
  pseudo_header->saddr = saddr;
  pseudo_header->daddr = daddr;
  pseudo_header->reserved = 0;
  pseudo_header->proto = proto; // IPPROTO_(TCP|UDP|ICMP)
  pseudo_header->len = htons(pkt_len);
  pseudo_header->data = pkt; // without copying pkt
}
uint16_t calc_checksum(const void *data, uint16_t len,
                       uint8_t proto) { // RFC 1071 [64]
  if (data == NULL) {
    printf("Data is NULL for calculating the check sum\n");
    return 0;
  }
  uint16_t offset_chk_sum = 0;
  switch (proto) {
  case IPPROTO_IP:
    offset_chk_sum = 10;
    break;
  case IPPROTO_TCP:
    offset_chk_sum = sizeof(pseudo_hdr) - sizeof(uint8_t *) + 16;
    break;
  case IPPROTO_UDP:
    offset_chk_sum = sizeof(pseudo_hdr) - sizeof(uint8_t *) + 6;
    break;
  case IPPROTO_ICMP:
    offset_chk_sum = sizeof(pseudo_hdr) - sizeof(uint8_t *) + 2;
    break;
  default:
    printf("Unknown protocol type\n");
    return 0;
  }
  register uint32_t sum = 0;
  const uint16_t *buff = (const uint16_t *)data;
  size_t i = 0;
  while (len > 1) {
    if ((proto == IPPROTO_TCP || proto == IPPROTO_UDP ||
         proto == IPPROTO_ICMP) &&
        i == sizeof(pseudo_hdr) - sizeof(uint8_t *))
      buff = (const uint16_t *)(((const pseudo_hdr *)data)->data);
    if (i != offset_chk_sum) { // skip field check sum
      sum += *buff;
      if ((sum & 0x80000000) != 0)
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    len -= 2;
    i += 2;
    ++buff;
  }
  if (len > 0)
    sum += *(u_char *)buff;
  while ((sum >> 16) != 0)
    sum = (sum & 0xFFFF) + (sum >> 16);
  return (uint16_t)(~sum);
}
```

А.10 Устойчивость СОА к стрессовым сетевым нагрузкам

В данном разделе кратко описана методика, которая позволяет повысить скорость обработки сетевого трафика и уменьшить число отбрасываемых пакетов. Тестирование стрессовой нагрузки производилось на примере Bro, которая запускалась с настройками по умолчанию на машине под управлением ОС Debian Jessie 8.4 со следующей конфигурацией: объем RAM 1536 MB, CPU Intel Core i5-2400 3140.366 MHz, размер кэша третьего уровня 6144 KB, число виртуальных процессоров 3.

На рисунке А.11 представлены графики, отражающие зависимость числа отброшенных пакетов от скорости анализируемого СОА трафика. Левый график (рисунок А.11(а)) соответствует процессу обработки всех пакетов на одном интерфейсе, правый график (рисунок А.11(б)) есть зависимость количества отброшенных СОА пакетов с применением процедуры балансировки по двум интерфейсам. При перенаправлении трафика на три интерфейса на максимальной скорости уже не наблюдалось отбрасывания пакетов.



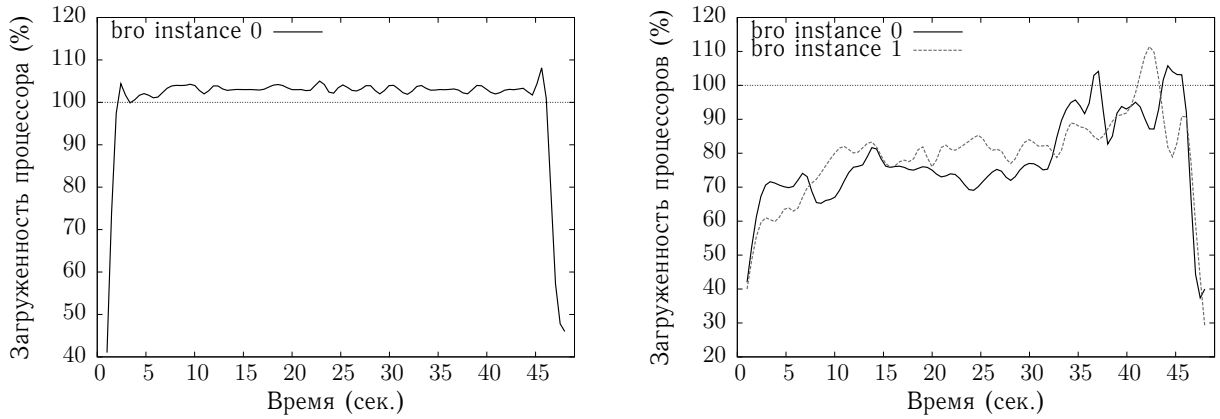
(а) для случая одного процесса Bro (б) для случая двух процессов Bro

Рисунок А.11 — Зависимость числа отброшенных пакетов от скорости обрабатываемого трафика

Эксперименты проводились с использованием pcap-файла, состоящего из 6 млн. пакетных записей, общим размером 426 MB. В результате применения методики балансировки трафика в случае двух интерфейсов доля принятых пакетов была разбита в отношении 50.62625% и 49.37375% соответственно для

первого и второго интерфейсов. Из рисунков видно, что процентное число отбрасываемых пакетов снизилось в сотни раз.

На рисунке А.12 представлены графики, отображающие загрузку CPU в зависимости от времени обработки приемного трафика. На левом графике (рисунок А.12(а)) показан ход загрузки процессора одним экземпляром СОА с частотой в 1 секунду. Правый график (рисунок А.12(б)) содержит те же сведения для случая двух процессов СОА. Данные, изображенные на этих графиках, относятся к скорости трафика 130 килопакетов в секунду. Замеры производились при помощи инструмента pidstat из набора утилит sysstat.

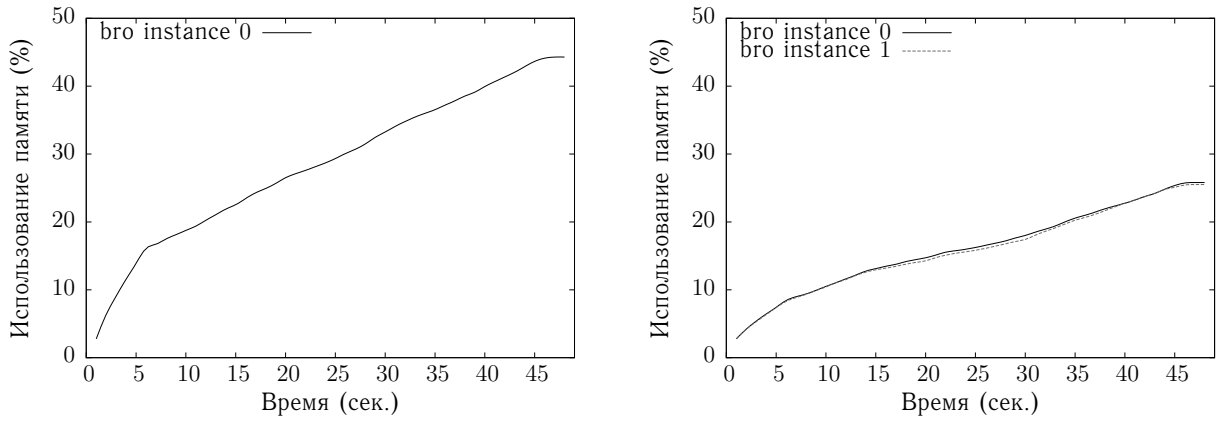


(а) для случая одного процесса Bro (б) для случая двух процессов Bro

Рисунок А.12 — Зависимость загрузки центральных процессоров от времени обработки трафика

В случае запуска одного экземпляра СОА загруженность процессора бóльшую часть времени не опускается ниже 100% и в среднем составляет 99.5408%. При использовании нескольких копий СОА среднее потребление процессорного времени занимает 77.3125% и 78.7083% соответственно первым и вторым процессом СОА, и только в отдельные моменты времени загруженность процессора превышает отметку в 100%.

На рисунке А.13 представлены графики, отображающие зависимость использования памяти от времени обработки трафика для каждого процесса СОА. Левый график (рисунок А.13(а)) соответствует случаю обработки всех пакетов одним процессом СОА, правый график (рисунок А.13(б)) — двумя процессами СОА. Как и в предыдущем примере, данные были получены в результате обработки трафика на скорости 130 килопакетов в секунду.



(а) для случая одного процесса Bro (б) для случая двух процессов Bro

Рисунок А.13 — Зависимость использования оперативной памяти от времени обработки трафика

Из рисунков можно отчетливо проследить, что до некоторого порогового момента наблюдается линейная зависимость размера потребляемой памяти с течением времени. Причем порядок роста потребляемой памяти в первом случае для одного запущенного процесса почти в два раза превышает порядок роста используемой памяти для каждого отдельно взятого процесса во втором случае. Отсюда можно сделать вывод, что применение методики балансировки трафика с одновременным запуском нескольких копий СОА (без учета объема потребляемой самим балансировщиком памяти) не приводит к существенным затратам по объему используемой памяти в сравнении с запуском одной копии СОА.

На рисунке А.14 представлен механизм выделения памяти и хранения хешированных записей о сетевых соединениях в пакетном балансировщике. На этом рисунке представлены два объекта: пул свободных записей (free_ip_conns) и таблица используемых записей (ip_conn_table). Когда таблица ip_conn_table пуста, выполняется free_ip_conns=ip_conn_pool. Роль первого объекта заключается в размещении незанятых записей, готовых к добавлению и дальнейшему поиску в таблице используемых записей. Перед выполнением основных функций пакетной балансировки производится предварительное выделение памяти размером 3 млн. записей (S). Все записи в пуле объединяются друг с другом посредством простого механизма односторонних ссылок на соседний элемент. Второй объект — это массив, каждый элемент которого проиндексирован значением хеш-функции fnv и представлен как однонаправленный связный список.

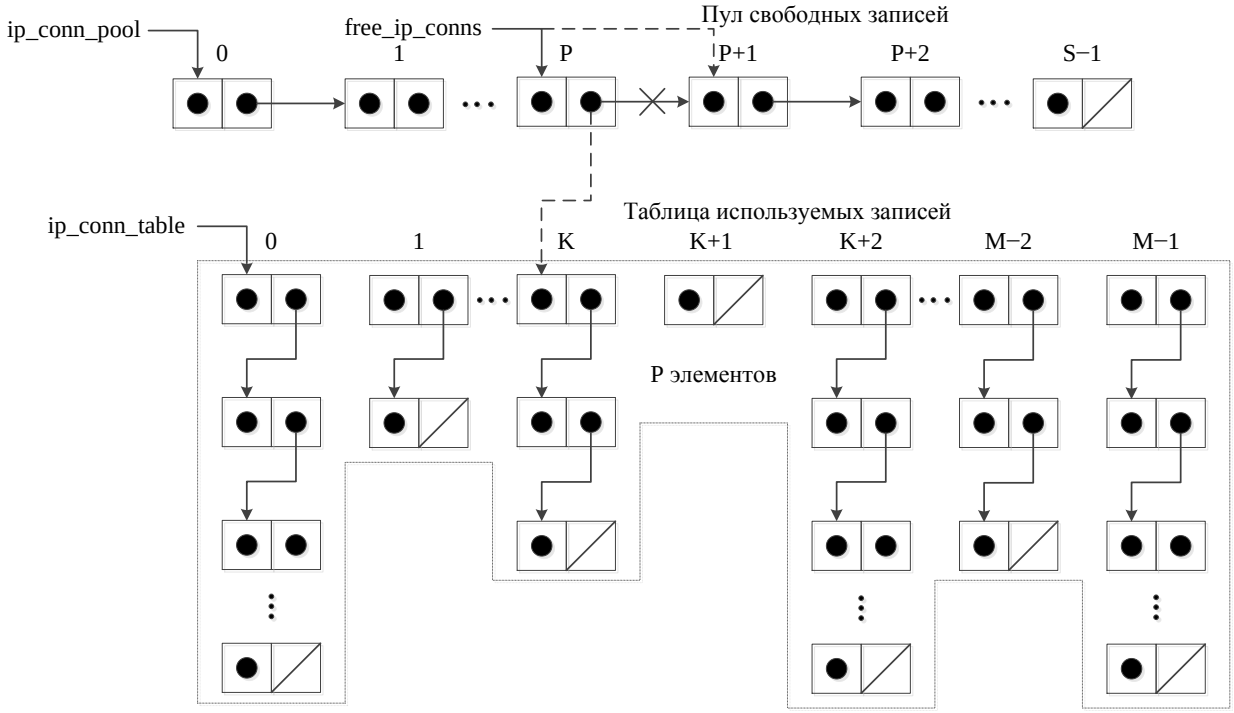


Рисунок А.14 — Механизм выделения памяти и хранения хешированных записей о сетевых соединениях в пакетном балансировщике

В экспериментах размер таблицы (M) был выбран равным 1 млн. Аргументом выбранной хеш-функции является неупорядоченная пара IP-адресов отправителя и приемника. При необходимости добавления в таблицу нового элемента, описывающего соединение между хостами с IP-адресами A и B, указатель на текущий головной P-ый элемент пула (free_ip_conns) перемещается на следующий (P+1)-ый элемент, а новый выделенный элемент помещается в начало списка ip_conn_table[K], где K=fnv(A,B)%M.

На рисунке А.15 представлена блок-схема функционирования пакетного балансировщика. Особенностью алгоритма является то, что пакетные данные, передаваемые между парой фиксированных IP-адресов, всегда попадают в один и тот же интерфейс. Т.с. сохраняется важный механизм stateful inspection, характерный для тех СОА, которые поддерживают обнаружение сетевых нарушений как на уровне отдельных пакетов, так и на уровне сессий. Одним из очевидных недостатков предложенного алгоритма является необходимость двойного вычисления хеш-функции fnv для пары IP-адресов. Этот изъян устранен в реализации сетевого анализатора, функционирование которого описано в разделе 3.2.

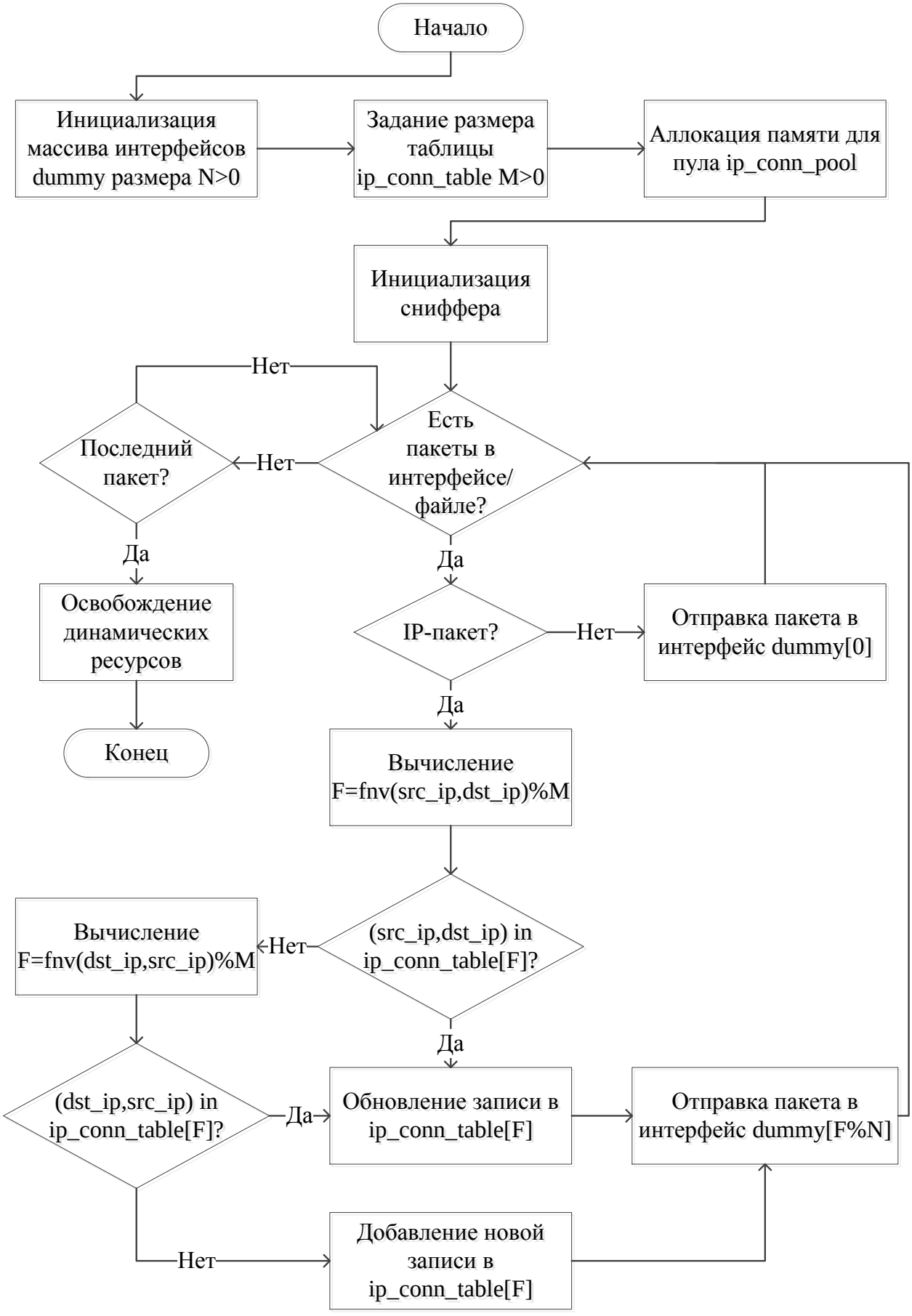


Рисунок А.15 — Алгоритм функционирования пакетного балансировщика

Выводы по приложению А

В данном приложении проведен обзор пяти свободно распространяемых СОА (Snort, Suricata, Bro, OSSEC и Prelude), которые на данный момент активно развиваются, имеют широкую популярность в научно-исследовательском и инженерном сообществах. Для каждой из рассмотренных СОА представлены ее архитектура и основные механизмы функционирования. Предложены несколько программных способов для повышения производительности СОА.

Представлены результаты ряда экспериментов, нацеленных на проверку сетевых СОА Snort, Suricata и Bro, а именно их способности к обнаружению атак со скрытием и со вставкой. Предложены 25 сценариев компрометирующих последовательностей сетевых пакетов, которые направлены на выявление недостатков в реализации сетевого анализатора каждой СОА и несоответствия его внутреннего функционирования работе сетевого стека ОС на защищаемом хосте. В роли используемых ОС рассматривались Linux с версией ядра 3.16.0-4 и Windows 7. Результаты экспериментов показали, что ни одна из представленных сигнатурных сетевых СОА не имеет показателя числа пропусков атаки (false negative), равного нулю.

Предложена методика балансировки сетевого трафика с целью распределения нагрузки между несколькими независимо исполняемыми на одной машине экземплярами СОА. Суть методики заключается в выполнении нескольких шагов: перенаправлении трафика с одного внешнего сетевого интерфейса на несколько внутренних сетевых интерфейсов и одновременном функционировании нескольких копий СОА, каждая из которых прослушивает определенный внутренний интерфейс. Сетевой поток дизъюнктно разбивается таким образом, чтобы у каждого запущенного процесса СОА сохранялась возможность анализа всех пакетов в рамках захваченной сессии. На примере Bro продемонстрированы результаты применения этого подхода, которые доказали его высокую эффективность на основе измерения числа отбрасываемых пакетов.

Приложение Б

Примеры скриптов обнаружения атаки «Brute Force»

Ниже представлены примеры правил обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами Snort, Suricata, Bro, OSSEC и Prelude. Предупреждение о возможной атаке выдается в случае, если число попыток неуспешной аутентификации на сервере 10.0.1.101 (порт 3306) превышает 10 за одну минуту с любого компьютера подсети 10.0.1.*.

Листинг Б.1 Правило обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами Snort

```
# login_failed.rules
alert tcp 10.0.1.101 3306 -> 10.0.1.0/24 any ( \
 content:"Access denied for user"; \
 msg:\
 "MySQL server: too much unsuccessful authentication attempts"; \
 flow:established,from_server; \
 detection_filter:track by_src, count 10, seconds 60; \
 sid:1000001; \
 rev:1)
```

Листинг Б.2 Правило обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами Suricata

```
# login_failed.rules
alert tcp 10.0.1.101 3306 -> 10.0.1.0/24 any ( \
 msg:\
 "MySQL server: too much unsuccessful authentication attempts"; \
 luajit:login_failed.lua; sid:1000002; \
 rev:1;)

# login_failed.lua
failed_attempts = {}

function init(args)
  local needs = {["packet"] = tostring(true),
```

```
                ["payload"] = tostring(true)}
  return needs
end

function match(args)
  if (string.match(args["payload"],
    "Access denied for user") == nil) then
    return 0
  end
  eth_hdr_len = 14
  ip_hdr_len = bit.band(args["packet"]:
               byte(eth_hdr_len + 1, eth_hdr_len + 1), 0x0F) * 4
  if (ip_hdr_len < 20) then
    return 0
  end
  src_ip, dst_ip = "", ""
  for i=0,3 do
    if src_ip ~= "" then
      src_ip = src_ip .. "."
      dst_ip = dst_ip .. "."
    end
    src_ip = src_ip .. tostring(args["packet"]:
                        byte(eth_hdr_len + 13 + i,
                             eth_hdr_len + 13 + i))
    dst_ip = dst_ip .. tostring(args["packet"]:
                        byte(eth_hdr_len + 17 + i,
                             eth_hdr_len + 17 + i))
  end
  cur_time = os.time()
  for i, v in pairs(failed_attempts) do
    t = {}
    for _, a in pairs(v) do
      if (cur_time - a <= 60) then
        table.insert(t, a)
      end
    end
    failed_attempts[i] = t
  end
```

```
  if (failed_attempts[dst_ip] == nil) then
    failed_attempts[dst_ip] = {}
  end
  table.insert(failed_attempts[dst_ip], cur_time)
  if (table.getn(failed_attempts[dst_ip]) > 10) then
    print("MySQL server: too much unsuccessful " ..
          "authentication attempts from " .. dst_ip)
    return 1
  end
  return 0
end

return 0
```

Листинг Б.3 Правило обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами Bro

```
# login_failed.bro
type mysql_login_counter: record
  {
  n_summary_failed_auth_logins: count &default = 0;
  timestamps: vector of time;
  };

global g_mysql_n_logins: table[addr] of mysql_login_counter;

global g_msg =
  "MySQL server: too much unsuccessful authentication attempts";

const duration_secs = 60;

const max_failed_attempts = 10;

function is_addr_in_10_0_1(a: addr): bool
  {
  local s: subnet = 10.0.1.0/24;
  return (a in s);
  }
```

```
event connection_established(c: connection)
  {
  local l4_proto = get_port_transport_proto(c$id$orig_p);
  local src_addr = c$id$orig_h;
  if (l4_proto == tcp && fmt("%d", c$id$resp_p) == "3306" &&
      fmt("%s", c$id$resp_h) == "10.0.1.101" &&
      is_addr_in_10_0_1(src_addr))
    {
    if (src_addr !in g_mysql_n_logins)
      {
        g_mysql_n_logins[src_addr] =
          [$timestamps = vector()];
      }
    }
  }

function is_pkt_contain_failed_login_string(s: string): bool
  {
  return (strstr(s, "Access denied for user") != 0) ? T : F;
  }

function remove_expired_logins_by_addr(a: addr)
  {
  local ts = current_time();
  local v: vector of time = vector();
  for (n in g_mysql_n_logins[a]$timestamps)
    {
    local diff_dur = ts - g_mysql_n_logins[a]$timestamps[n];
    if (double_to_count(interval_to_double(diff_dur)) <=
        duration_secs)
      {
      v[|v|] = g_mysql_n_logins[a]$timestamps[n];
      }
    }
  g_mysql_n_logins[a]$timestamps = v;
  }

event tcp_packet(c: connection, is_orig: bool, flags: string,
```

```
                seq: count, ack: count, len: count,
                payload: string)
  {
  for (a in g_mysql_n_logins)
    {
    remove_expired_logins_by_addr(a);
    if (a == c$id$orig_h &&
        is_pkt_contain_failed_login_string(payload))
      {
      ++g_mysql_n_logins[a]$n_summary_failed_auth_logins;
      local n_timestamps = |g_mysql_n_logins[a]$timestamps|;
      g_mysql_n_logins[a]$timestamps[n_timestamps] =
        current_time();
      if (++n_timestamps > max_failed_attempts)
        {
        print fmt("%s from %s (%d secs: %d/all: %d)", g_msg,
          a, duration_secs, n_timestamps,
          g_mysql_n_logins[a]$n_summary_failed_auth_logins);
        }
      }
    }
  }
```

Листинг Б.4 Правило обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами OSSEC

```
<!-- /var/ossec/etc/ossec.conf -->
<ossec_config>
  <global>
    <email_notification>no<email_notification>
  </global>
  <rules>
    <include>mysql_rules.xml</include>
  </rules>
  <localfile>
    <log_format>mysql_log</log_format>
    <location>/var/log/mysql/error.log</location>
  </localfile>
</ossec_config>
```

```
15
<!-- /var/ossec/etc/decoder.xml -->
<decoder name="mysql_log">
  <prematch>^MySQL log:</prematch>
</decoder>
20
<!-- /var/ossec/rules/mysql_rules.xml -->
<group name="mysql_log,">
  <rule id="50100" level="0">
    <decoded_as>mysql_log</decoded_as>
25  <description>MySQL messages grouped.</description>
  </rule>
  <rule id="50181" level="1">
    <if_sid>50100</if_sid>
    <match>Access denied for user</match>
30  <regex>
      [Warning] Access denied for user '\S+'@'10.0.1.\.*'
    </regex>
    <description>
      MySQL: unsuccessful authentication attempt.
35  </description>
    <group>authentication_failed,</group>
  </rule>
  <rule id="50182" level="10" frequency="10"
   timeframe="60" ignore="1">
40  <if_matched_sid>50181</if_matched_sid>
    <description>
      MySQL: many unsuccessful authentication attempts.
    </description>
    <group>authentication_failed,</group>
45  </rule>
</group>
```

Листинг Б.5 Правило обнаружения атаки „агрессивного“ подбора пароля для получения доступа к MySQL-серверу средствами Prelude

```
# brute_force_detection_sensor.pl
#!/usr/bin/perl -w
use strict;
```

```
use Getopt::Long;
use constant DEFAULT_PORT => 3306;
use constant DEFAULT_ADDR => '10.0.1.101';
use constant DEFAULT_IFACE => 'eth1';

START: main();

sub main
{
  my ($show_help_msg, $port, $addr, $iface) =
    (0, DEFAULT_PORT, DEFAULT_ADDR, DEFAULT_IFACE);
  my $usage_info = "Usage: $0 (Branitskiy Alexander)\n" .
    "\t--help|-h|-?               show help info and exit\n" .
    "\t--port=<port>|-p <port>    set listen port <port>\n" .
    "\t--addr=<addr>|-a <addr>    set listen address <addr>\n" .
    "\t--iface=<iface>|-i <iface> set listen interface <iface>\n";
  GetOptions("help|h|?" => \$show_help_msg,
             "port|p=i" => \$port,
             "addr|a=s" => \$addr,
             "iface|i=s" => \$iface) ||
    die $usage_info;
  die $usage_info if ($show_help_msg);
  my $login_failed_obj =
    new BruteForceDetection({ PORT => $port,
                              ADDR => $addr,
                              IFACE => $iface });
  $login_failed_obj->start();
}

package BruteForceDetection;
use Net::Pcap;
use Socket;
use PreludeEasy;
use constant ETH_HLEN => 14;
use constant ETHERTYPE_IP => 0x0800;
use constant IPPROTO_TCP => 6;
use constant MAX_PKT_LEN => 65535;
```

```
sub new
{
  my ($class, $args) = @_;
  if (ref($args) ne "HASH" || keys(%{$args}) !~ m/[23]/ ||
      !defined($args->{PORT} || !defined($args->{ADDR})))
  {
    die "BruteForceDetection: incorrect input arguments\n";
  }
  my $self = { PORT => $args->{PORT},
               ADDR => $args->{ADDR},
               IFACE => $args->{IFACE},
               FLOGINS => {},
               CLNT => new PreludeEasy::ClientEasy('pl_sensor') };
  # start Prelude client
  $self->{CLNT}->Start();
  bless $self, $class;
  return $self;
}

sub start
{
  my ($self, $err) = (shift, undef);
  unless (defined($self->{IFACE}))
  {
    $self->{IFACE} = Net::Pcap::lookupdev(\$err);
    defined($err) && die "Could not find device\n";
  }
  # open device in promiscuous mode
  my $pcap = Net::Pcap::open_live($self->{IFACE}, MAX_PKT_LEN, 1,
                                  0, \$err);
  if (!defined($pcap) || defined($err))
  {
    die "Could not find device\n";
  }
  my ($filter, $filter_str, $optimize, $mask) =
    (undef, "ether proto " . ETHERTYPE_IP . " and ip proto " .
     IPPROTO_TCP . " and src host $self->{ADDR} and src port " .
     "$self->{PORT} and dst net 10.0.1.0/24", 1, 0);
```

```
  Net::Pcap::compile($pcap, \$filter, $filter_str, $optimize,
                     $mask) == 0 ||
                     die "Could not compile filter\n";
  # set filter for capturing packets only from 10.0.1.*
  Net::Pcap::setfilter($pcap, $filter);
  # set SIGINT handler
  $SIG{INT} = sub { print "break loop\n";
                    Net::Pcap::breakloop($pcap); };
  # set callback function for processing every captured packet
  Net::Pcap::loop($pcap, -1, \&process_packet, $self);
  # close device
  Net::Pcap::close($pcap);
}

sub process_packet
{
  my ($self, $header, $packet) = @_;
  return if ($header->{caplen} < ETH_HLEN);
  my $to_int = sub { my ($str, $beg, $len) = @_;
                     return hex(unpack "H*",
                                substr($str, $beg, $len)); };
  my $eth_type = &$to_int($packet, 12, 2);
  return if ($eth_type != ETHERTYPE_IP ||
             $header->{caplen} < ETH_HLEN + 20 ||
             &$to_int($packet, ETH_HLEN + 9, 1) != IPPROTO_TCP);
  my ($ip_hdr_len, $ip_len) =
    ((((&$to_int($packet, ETH_HLEN, 1)) & 0x0F) << 2),
     &$to_int($packet, ETH_HLEN + 2, 2));
  # skip too small packet
  return if ($ip_hdr_len < 20 || $ip_len < $ip_hdr_len + 20 ||
    $header->{caplen} < ETH_HLEN + $ip_hdr_len + 20);
  my ($tcp_hdr_len, $tcp_len) =
    (((&$to_int($packet, ETH_HLEN + $ip_hdr_len + 12, 1) & 0xF0)
      >> 2), $ip_len - $ip_hdr_len);
  # skip too small packet
  return if ($ip_len < $ip_hdr_len + $tcp_hdr_len ||
             $header->{caplen} < ETH_HLEN + $ip_len);
  my $payload_len = $tcp_len - $tcp_hdr_len;
```

```
  return if ($payload_len < length("Access denied for user"));
  if (substr($packet, ETH_HLEN + $ip_hdr_len + $tcp_hdr_len) =~
      m/Access denied for user/)
  {
    my ($dst_ip_addr, $cur_time) =
      (inet_ntoa(pack("N",
                      &$to_int($packet, ETH_HLEN + 16, 4))),
       time());
    foreach my $key (keys(%{$self->{FLOGINS}}))
    {
      @{$self->{FLOGINS}->{$key}} =
        grep { $cur_time - $_ <= 60 } @{$self->{FLOGINS}->{$key}};
    }
    unless (exists($self->{FLOGINS}->{$dst_ip_addr}))
    {
      $self->{FLOGINS}->{$dst_ip_addr} = [$cur_time];
    }
    else
    {
      push @{$self->{FLOGINS}->{$dst_ip_addr}}, $cur_time;
    }
    if (scalar(@{$self->{FLOGINS}->{$dst_ip_addr}}) > 10)
    {
      print "BRUTE FORCE ATTACK FROM $dst_ip_addr\n";
      # create IDMEF message
      my $idmef = new PreludeEasy::IDMEF();
      # set fields of IDMEF message
      $idmef->Set("alert.additional_data(0).data",
                  "MySQL server: too much unsuccessful " .
                  "authentication attempts from $dst_ip_addr");
      # send IDMEF message
      $self->{CLNT}->SendIDMEF($idmef);
    }
  }
}
```

Приложение В

Грамматика интерпретатора интеллектуального ядра классификации объектов

Грамматика интерпретатора интеллектуального ядра классификации объектов является контекстно-свободной, использует в правилах только левую рекурсию и представляется следующим образом:

$$IG = \langle V_N, V_T, S, P \rangle,$$

где V_N = (⟨*ClsTree*⟩, ⟨*Vars*⟩, ⟨*VarsList*⟩, ⟨*Cls*⟩, ⟨*ClassifierFields*⟩, ⟨*Clss*⟩, ⟨*Expr*⟩, ⟨*Token*⟩, ⟨*Concat*⟩, ⟨*Sum*⟩, ⟨*Prod*⟩, ⟨*Div*⟩, ⟨*Merge*⟩, ⟨*If*⟩, ⟨*Cond*⟩, ⟨*Op*⟩, ⟨*Tokens*⟩, ⟨*OutputDataF*⟩, ⟨*OutputDataE*⟩, ⟨*OutputDataT*⟩, ⟨*Rules*⟩, ⟨*Field*⟩, ⟨*FieldString*⟩, ⟨*FieldNumber*⟩, ⟨*String*⟩, ⟨*Number*⟩) — множество нетерминалов, V_T = (‘`classifier_tree`’, ‘`vars`’, ‘`name`’, ‘`identifier`’, ‘`so_library`’, ‘`multithread`’, ‘`bagging`’, ‘`sls_file`’, ‘`opt_file`’, ‘`prf_file`’, ‘`num_groups`’, ‘`input_dim`’, ‘`output_dim`’, ‘`input_data`’, ‘`output_data`’, ‘`nested_classifiers`’, ‘`ordinary_data`’, ‘`idx#`’, ‘`concat`’, ‘`sum`’, ‘`prod`’, ‘`div`’, ‘`merge`’, ‘`if`’, ‘`type`’, ‘`format`’, ‘`detector`’, ‘`index`’, ‘`rules`’, ‘`=`’, ‘`>`’, ‘`<`’, ‘`>=`’, ‘`<=`’ ‘`{`’, ‘`}`’, ‘`(`’, ‘`)`’, ‘`:`’, ‘`"`’, ‘`,`’, `[^ (){},\t\n:"]+`, `[0-9]+`) — множество терминалов, S = ⟨*ClsTree*⟩ — начальный нетерминал, $P : V_N \rightarrow (V_N \bigcup V_T)^+$ — правила следующего вида:

⟨*ClsTree*⟩ → ‘`classifier_tree`’ ‘`:`’ ‘`{`’ ⟨*Vars*⟩ ⟨*Cls*⟩ ‘`}`’
| ‘`classifier_tree`’ ‘`:`’ ‘`{`’ ⟨*Cls*⟩ ‘`}`’

⟨*Vars*⟩ → ‘`vars`’ ‘`:`’ ‘`{`’ ⟨*VarsList*⟩ ‘`}`’

⟨*VarsList*⟩ → ⟨*String*⟩ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’
| ⟨*VarsList*⟩ ⟨*String*⟩ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’

⟨*Cls*⟩ → ‘`classifier`’ ‘`:`’ ‘`{`’ ⟨*ClassifierFields*⟩ ‘`}`’

⟨*ClassifierFields*⟩ → ‘`name`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`identifier`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`so_library`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`multithread`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`bagging`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`sls_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`ops_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`prf_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`num_groups`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`input_dim`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`output_dim`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`input_data`’ ‘`:`’ ‘`{`’ ⟨*Exp*⟩ ‘`}`’
| ‘`output_data`’ ‘`:`’ ‘`{`’ ⟨*OutputDataF*⟩ ‘`}`’
| ‘`nested_classifiers`’ ‘`:`’ ‘`{`’ ⟨*Clss*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`name`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`identifier`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`so_library`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`multithread`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`bagging`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`sls_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`opt_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`prf_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`num_groups`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`input_dim`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`output_dim`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`input_data`’ ‘`:`’ ‘`{`’ ⟨*Exp*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`output_data`’ ‘`:`’ ‘`{`’ ⟨*OutputDataF*⟩ ‘`}`’
| ⟨*ClassifierFields*⟩ ‘`nested_classifiers`’ ‘`:`’ ‘`{`’ ⟨*Clss*⟩ ‘`}`’

⟨*Clss*⟩ → ⟨*Cls*⟩
| ⟨*Clss*⟩ ⟨*Cls*⟩

⟨*Expr*⟩ → '"' ⟨*Token*⟩ '"'

⟨*Token*⟩ → ⟨*Concat*⟩
| ⟨*Sum*⟩
| ⟨*Prod*⟩
| ⟨*Div*⟩
| ⟨*Merge*⟩
| ⟨*If*⟩
| 'ordinary_data'
| 'idx#' ⟨*Number*⟩
| ⟨*Number*⟩
| ⟨*String*⟩

⟨*Concat*⟩ → 'concat' '(' ⟨*Tokens*⟩ ')'

⟨*Sum*⟩ → 'sum' '(' ⟨*Tokens*⟩ ')'

⟨*Prod*⟩ → 'prod' '(' ⟨*Tokens*⟩ ')'

⟨*Div*⟩ → 'div' '(' ⟨*Tokens*⟩ ')'

⟨*Merge*⟩ → 'merge' '(' ⟨*Tokens*⟩ ')'

⟨*If*⟩ → 'if' '(' ⟨*Cond*⟩ ',' ⟨*Token*⟩ ',' ⟨*Token*⟩ ')'

⟨*Cond*⟩ → ⟨*Token*⟩ ⟨*Op*⟩ ⟨*Token*⟩

⟨*Op*⟩ → '='
| '>'
| '<'
| '>='
| '<='

⟨*Tokens*⟩ → ⟨*Token*⟩
| ⟨*Tokens*⟩ ',' ⟨*Token*⟩

⟨*OutputDataF*⟩ → ‘`type`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`type`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’ ‘`format`’ ‘`:`’ ‘`{`’ ⟨*OutputDataE*⟩ ‘`}`’
| ‘`format`’ ‘`:`’ ‘`{`’ ⟨*OutputDataE*⟩ ‘`}`’ ‘`type`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’

⟨*OutputDataE*⟩ → ‘`detector`’ ‘`:`’ ‘`{`’ ⟨*OutputDataT*⟩ ‘`}`’
| ⟨*OutputDataE*⟩ ‘`detector`’ ‘`:`’ ‘`{`’ ⟨*OutputDataT*⟩ ‘`}`’

⟨*OutputDataT*⟩ → ‘`name`’ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’
| ‘`index`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`aux_index`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ‘`rules`’ ‘`:`’ ‘`{`’ ⟨*Rules*⟩ ‘`}`’
| ‘`sls_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ‘`opt_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`name`’ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`index`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`aux_index`’ ‘`:`’ ‘`{`’ ⟨*FieldNumber*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`rules`’ ‘`:`’ ‘`{`’ ⟨*Rules*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`sls_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’
| ⟨*OutputDataT*⟩ ‘`opt_file`’ ‘`:`’ ‘`{`’ ⟨*FieldString*⟩ ‘`}`’

⟨*Rules*⟩ → ⟨*Field*⟩ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’
| ⟨*Rules*⟩ ⟨*Field*⟩ ‘`:`’ ‘`{`’ ⟨*Field*⟩ ‘`}`’

⟨*Field*⟩ → ⟨*FieldString*⟩
| ⟨*FieldNumber*⟩

⟨*FieldString*⟩ → ‘`"`’ ⟨*String*⟩ ‘`"`’

⟨*FieldNumber*⟩ → ‘`"`’ ⟨*Number*⟩ ‘`"`’

⟨*String*⟩ → `[^ (){},\t\n:"]+`

⟨*Number*⟩ → `[0-9]+`

Приложение Г

Результаты экспериментов (рисунки и таблицы)

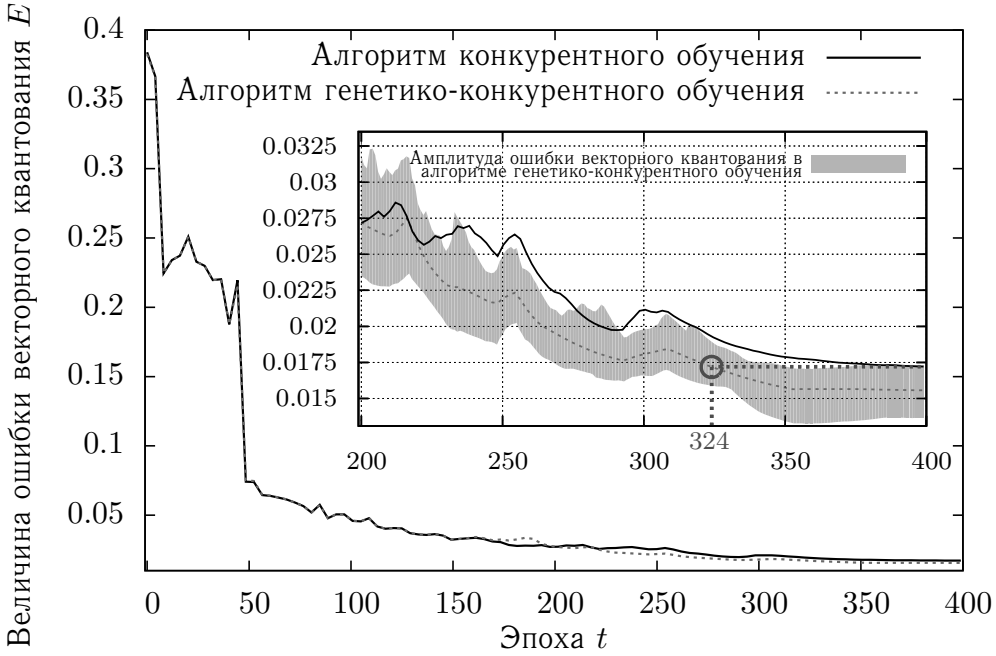


Рисунок Г.1 — Зависимость величины ошибки векторного квантования от номера эпохи алгоритма обучения для случая класса normal

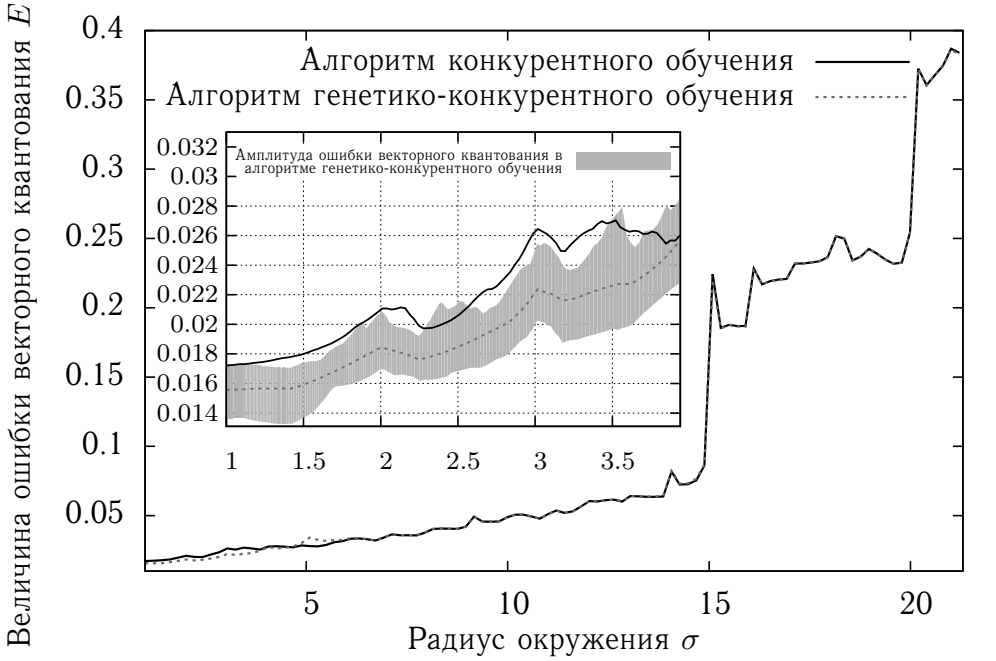


Рисунок Г.2 — Зависимость величины ошибки векторного квантования от радиуса окрестности нейрона-победителя для случая класса normal

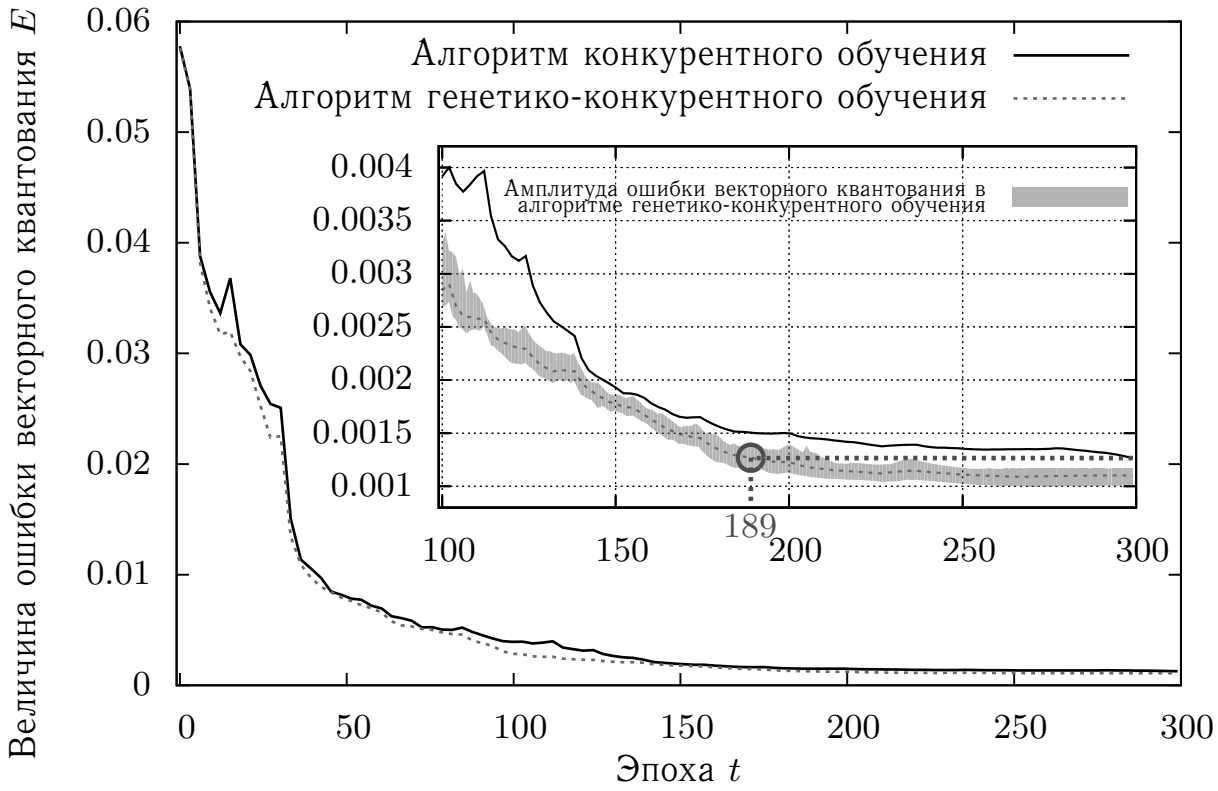


Рисунок Г.3 — Зависимость величины ошибки векторного квантования от номера эпохи алгоритма обучения для случая класса scan

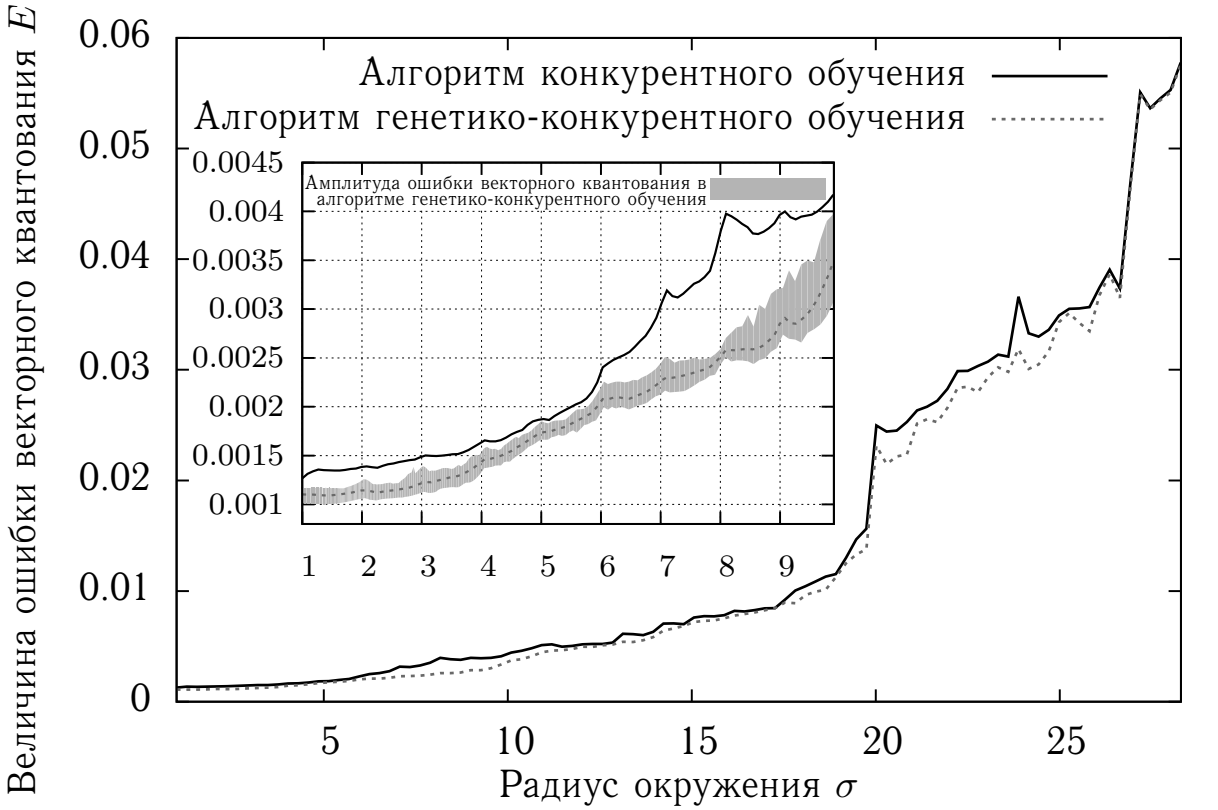


Рисунок Г.4 — Зависимость величины ошибки векторного квантования от радиуса окрестности нейрона-победителя для случая класса scan

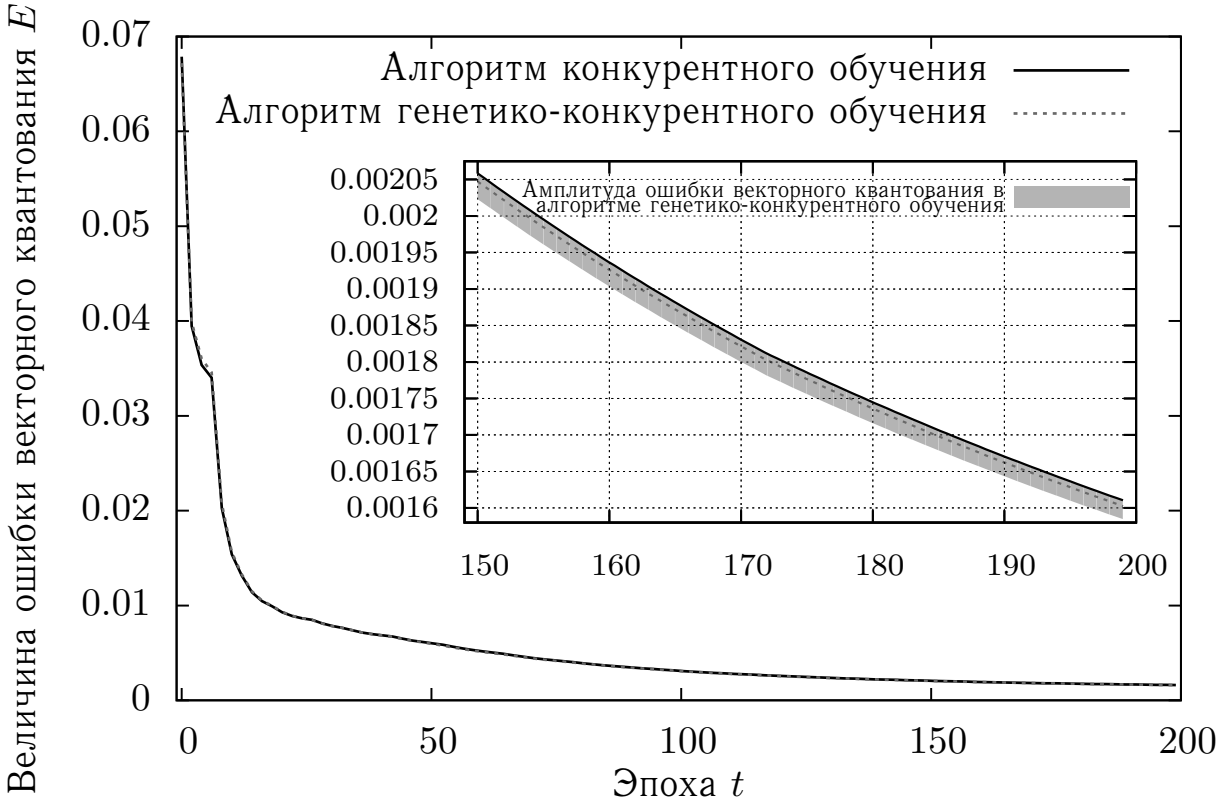


Рисунок Г.5 — Зависимость величины ошибки векторного квантования от номера эпохи алгоритма обучения для случая класса synflood

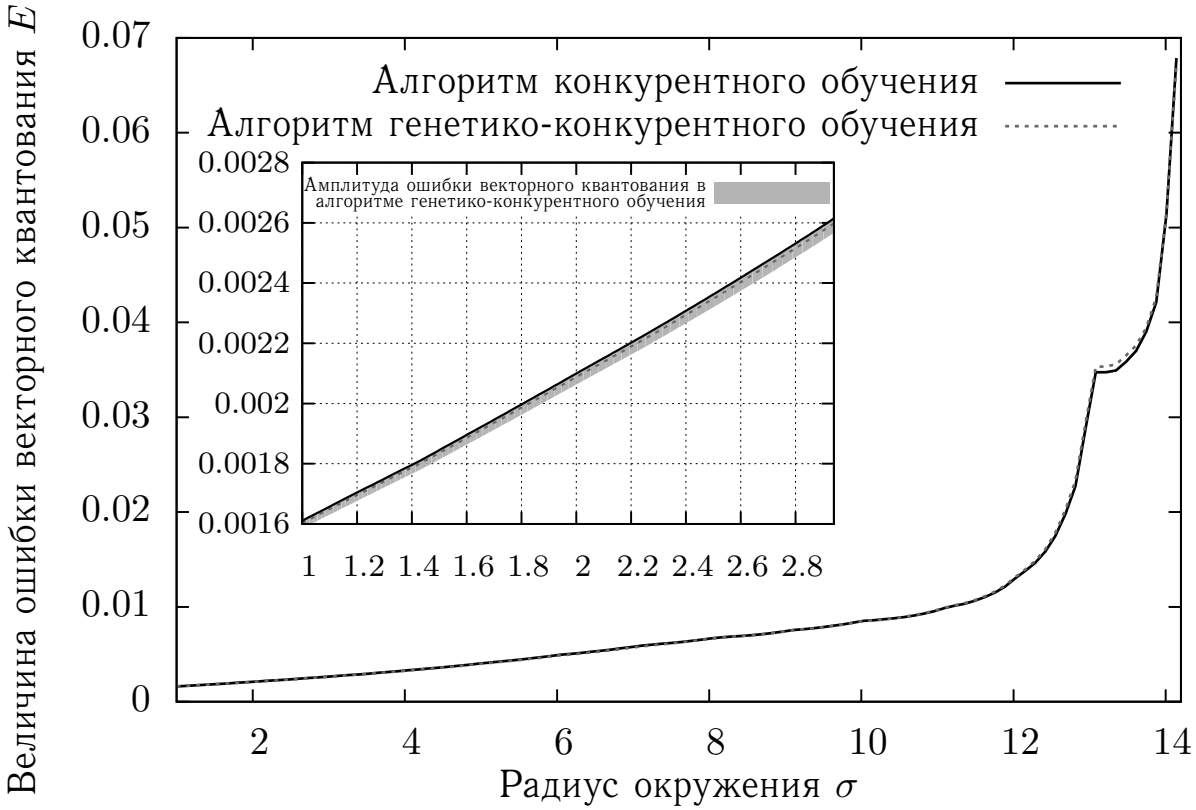


Рисунок Г.6 — Зависимость величины ошибки векторного квантования от радиуса окрестности нейрона-победителя для случая класса synflood

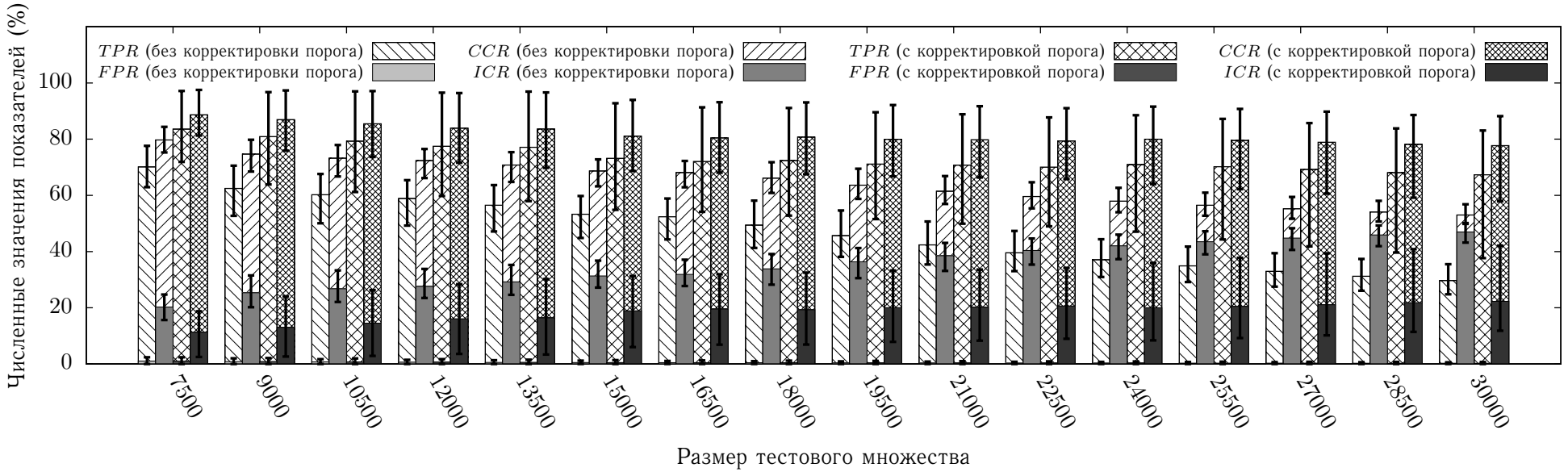


Рисунок Г.7 — Показатели эффективности TPR, FPR, CCR, ICR иммунной системы (6 детекторов 10–15 $\times$ 10–15)

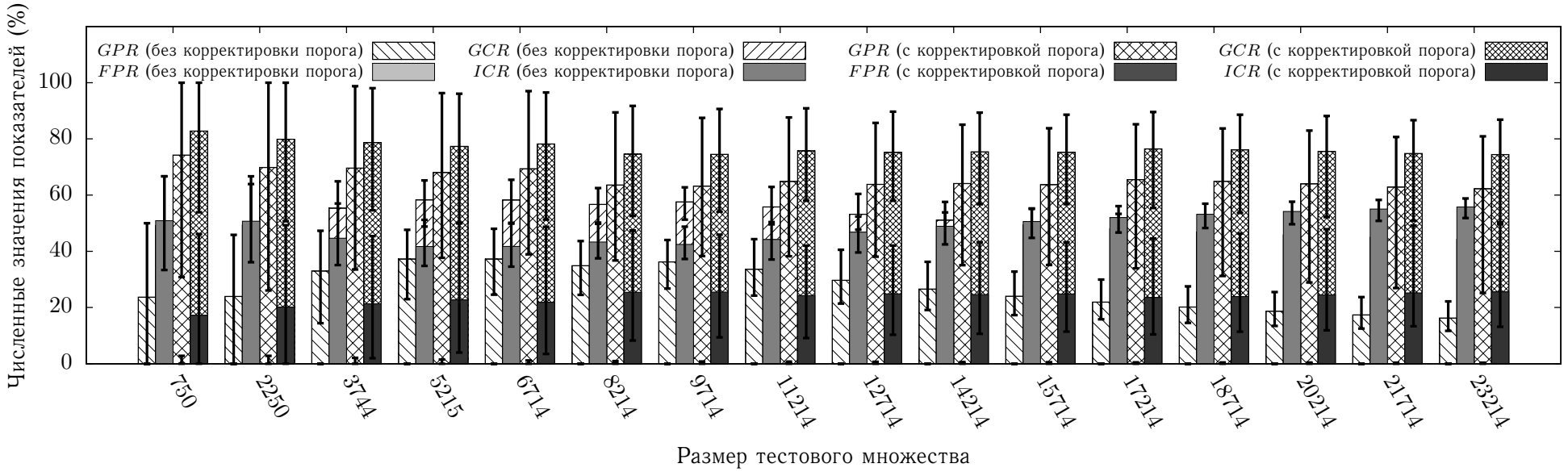


Рисунок Г.8 — Показатели эффективности GPR, FPR, GCR, ICR иммунной системы (6 детекторов 10–15 $\times$ 10–15)

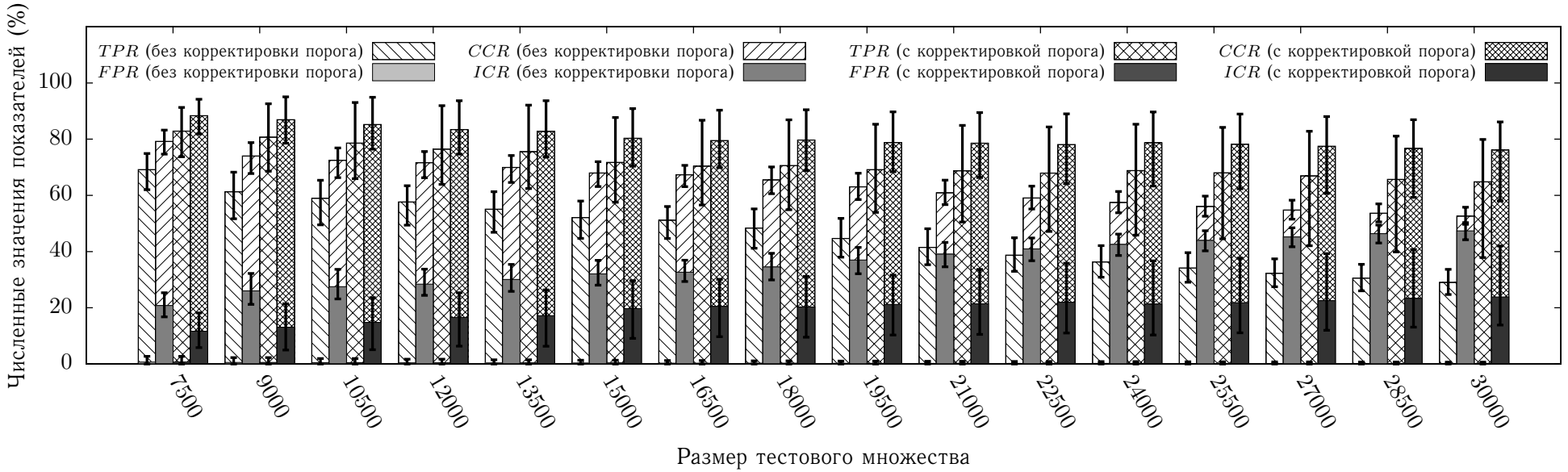


Рисунок Г.9 — Показатели эффективности TPR, FPR, CCR, ICR иммунной системы (6 детекторов 15–20 $\times$ 15–20)

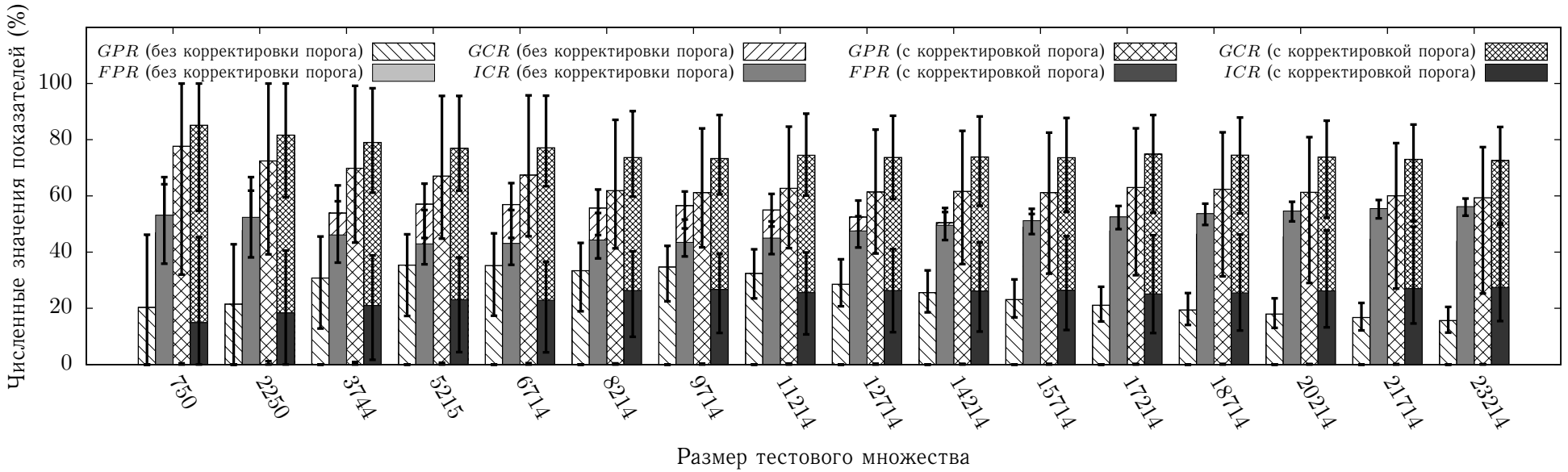


Рисунок Г.10 — Показатели эффективности GPR, FPR, GCR, ICR иммунной системы (6 детекторов 15–20 $\times$ 15–20)

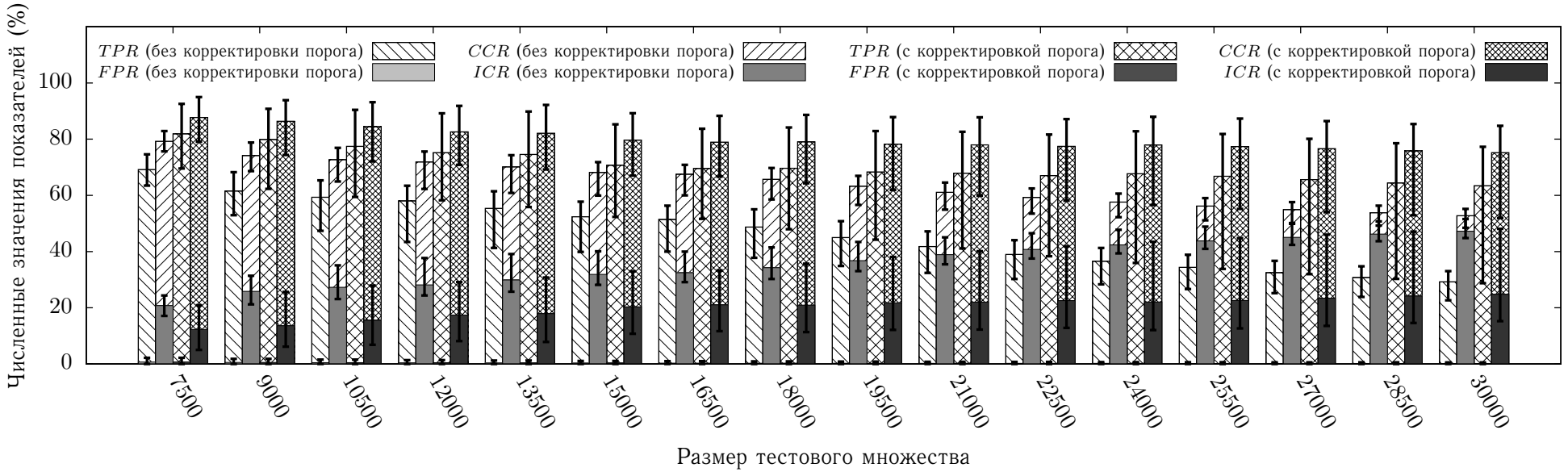


Рисунок Г.11 — Показатели эффективности TPR, FPR, CCR, ICR иммунной системы (6 детекторов 20–25 $\times$ 20–25)

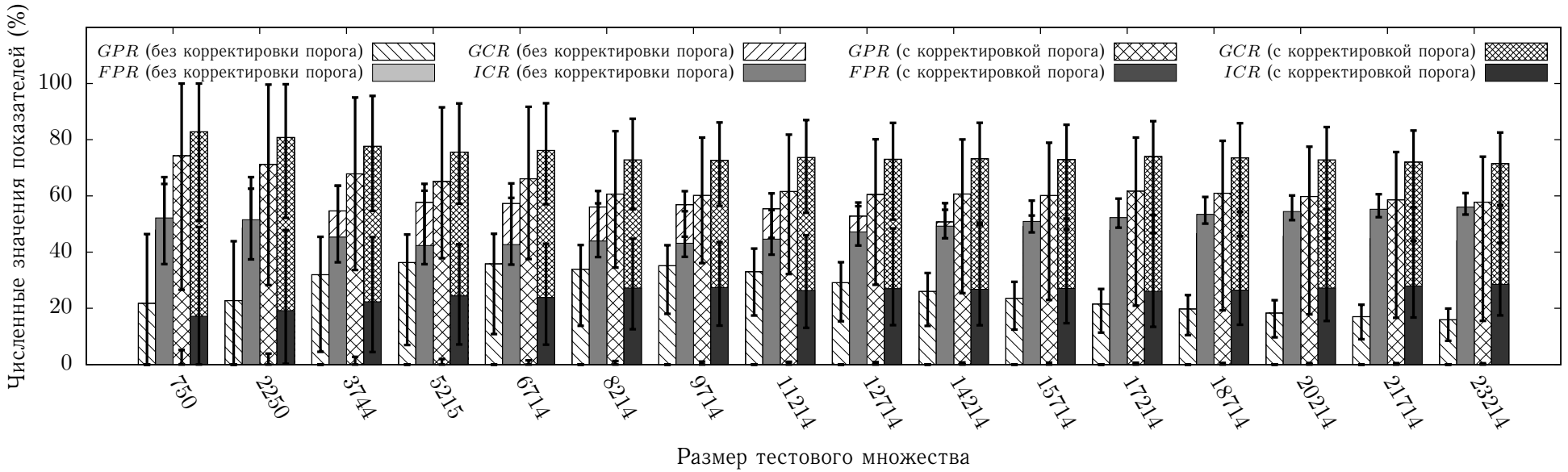


Рисунок Г.12 — Показатели эффективности GPR, FPR, GCR, ICR иммунной системы (6 детекторов 20–25 $\times$ 20–25)

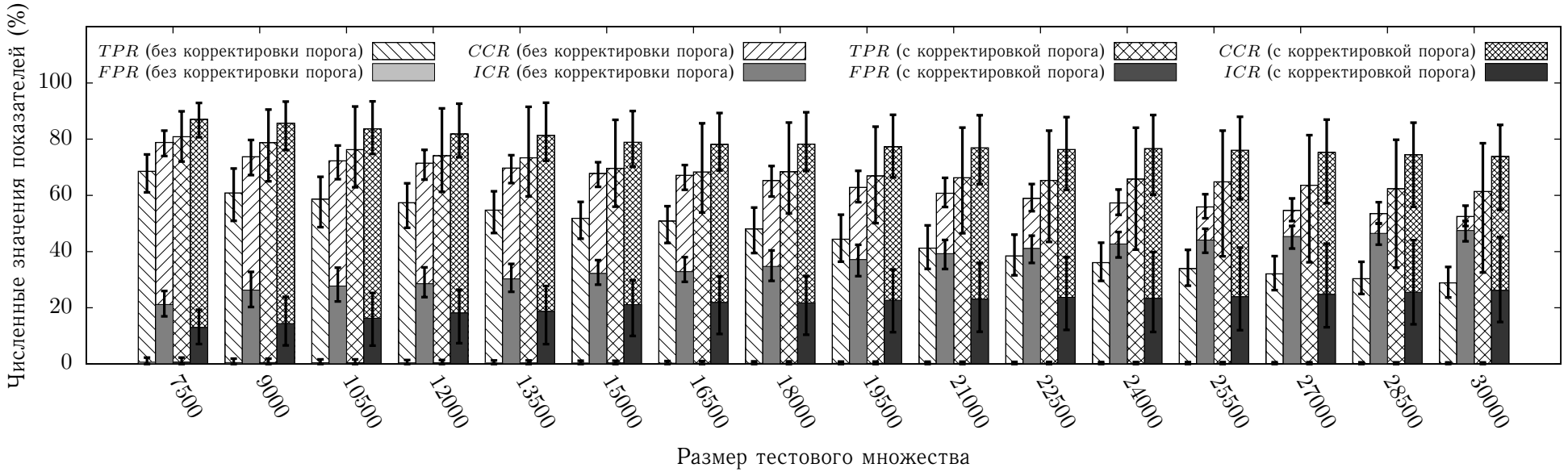


Рисунок Г.13 — Показатели эффективности TPR, FPR, CCR, ICR иммунной системы (6 детекторов 25–30 $\times$ 25–30)

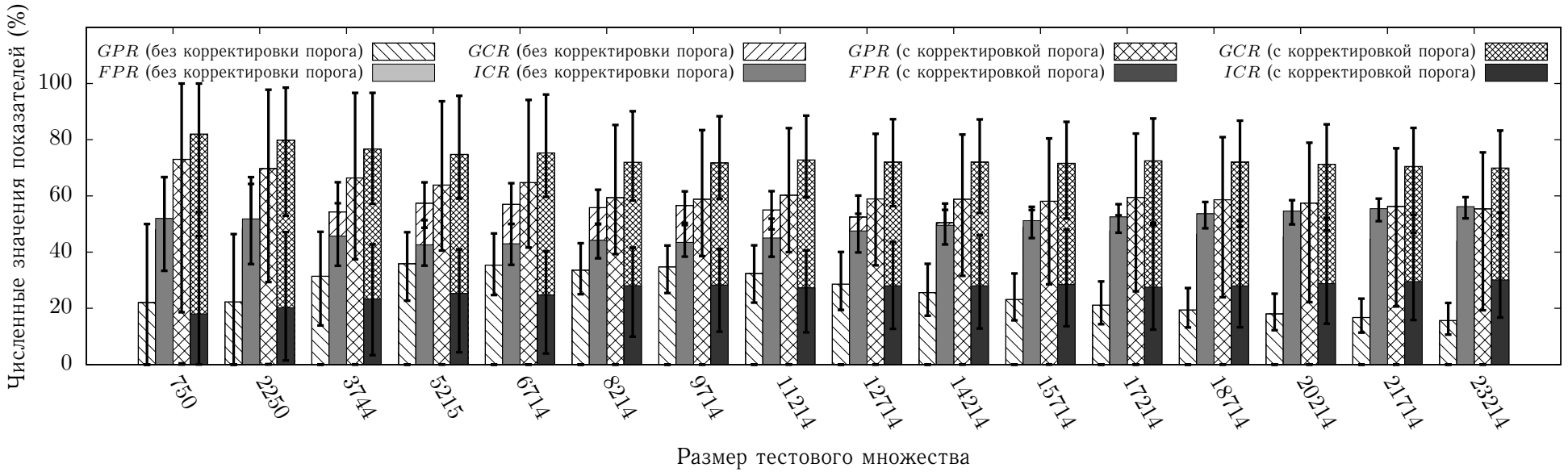


Рисунок Г.14 — Показатели эффективности GPR, FPR, GCR, ICR иммунной системы (6 детекторов 25–30 $\times$ 25–30)

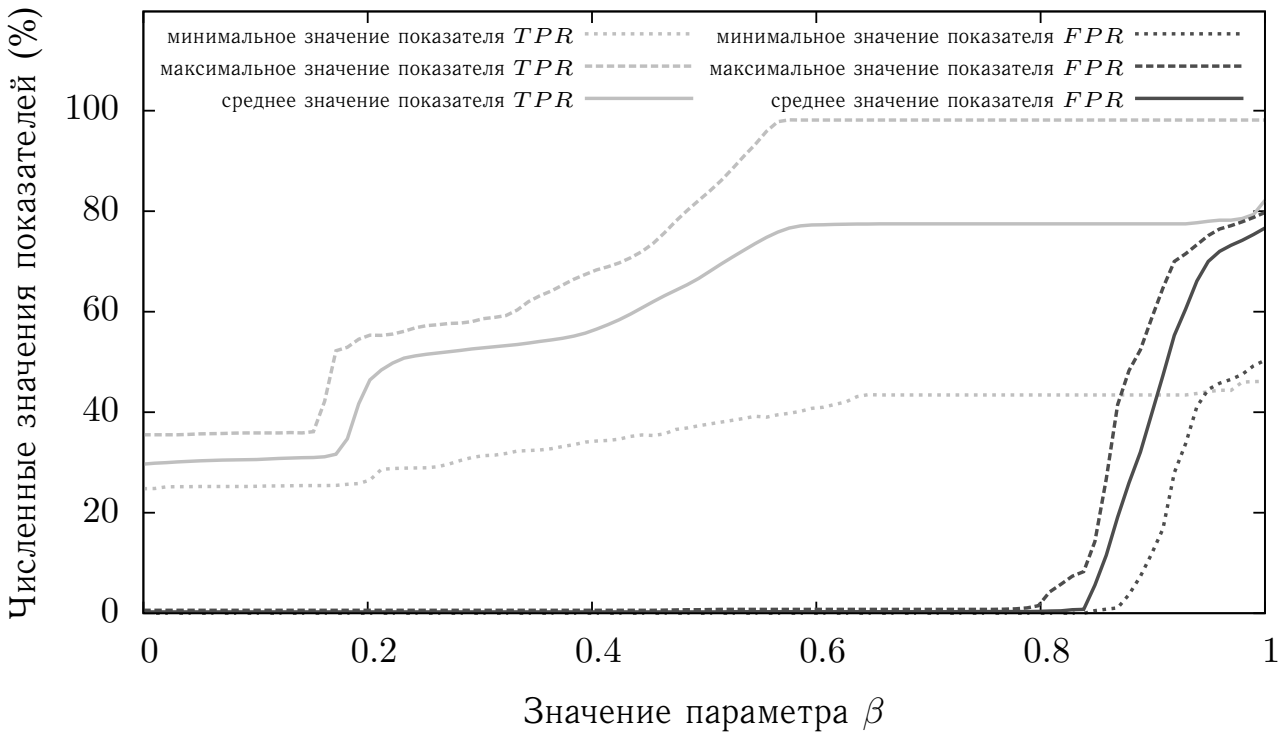


Рисунок Г.15 — Зависимость показателей TPR и FPR от значения параметра β на контрольном множестве из 30000 элементов (6 детекторов 10–15 $\times$ 10–15)

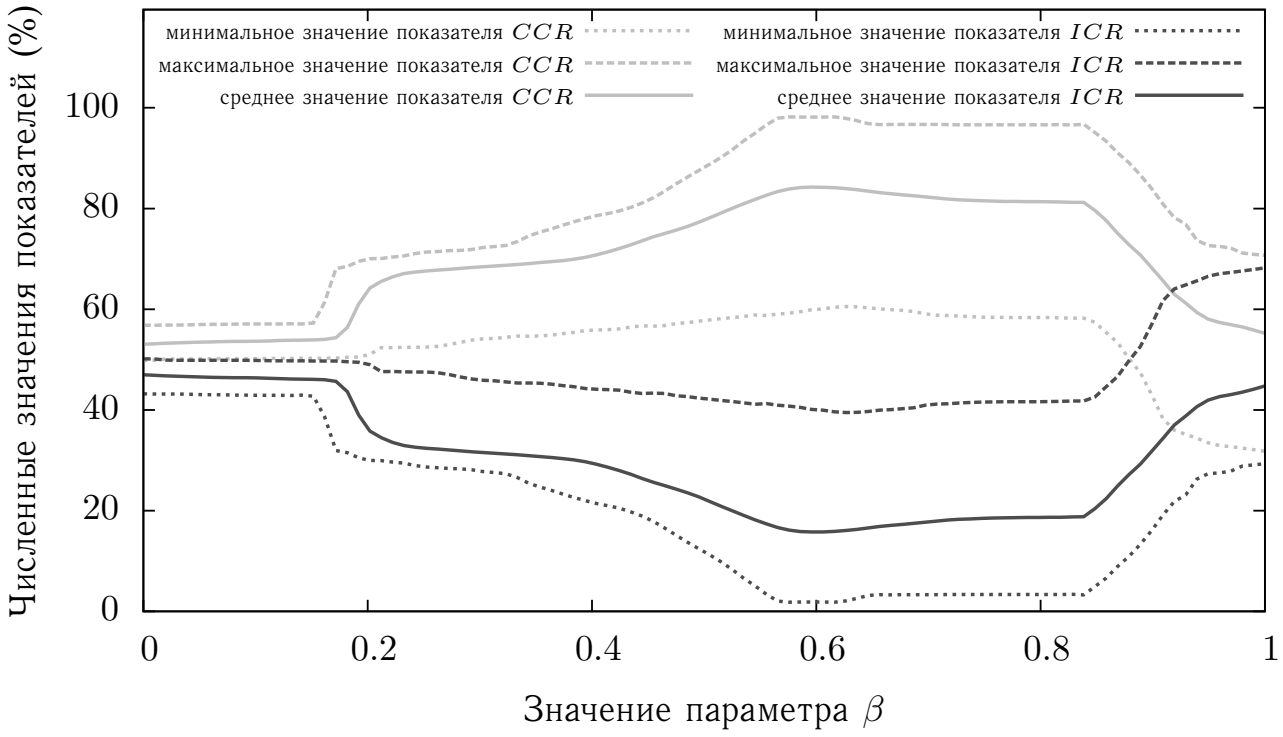


Рисунок Г.16 — Зависимость показателей CCR и ICR от значения параметра β на контрольном множестве из 30000 элементов (6 детекторов 10–15 $\times$ 10–15)

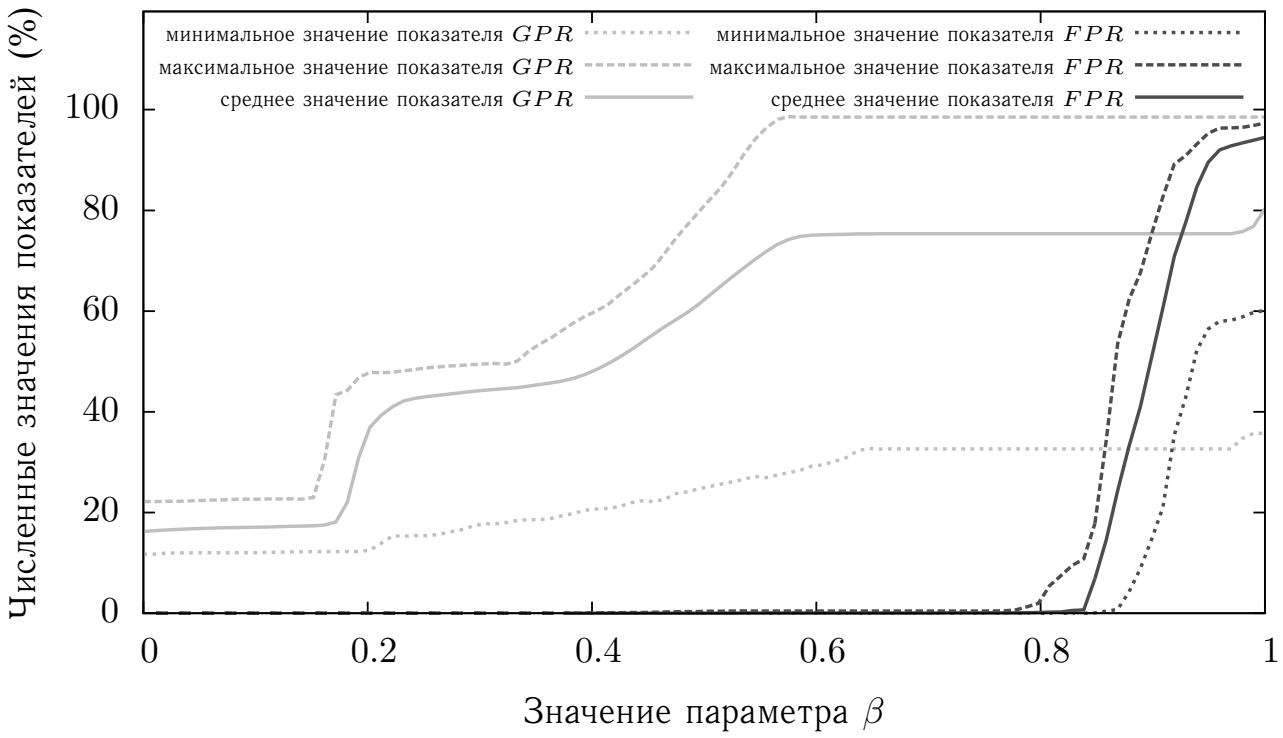


Рисунок Г.17 — Зависимость показателей GPR и FPR от значения параметра β на контрольном множестве из 23214 элементов (6 детекторов 10–15 $\times$ 10–15)

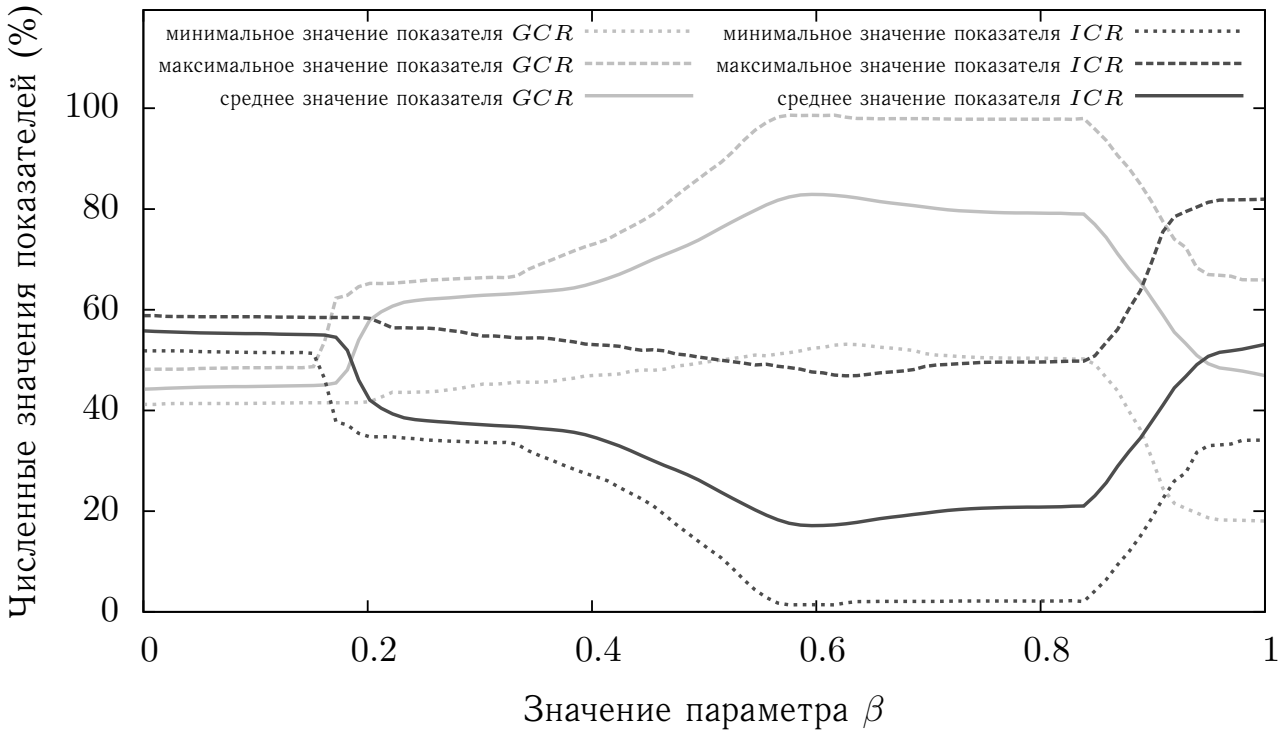


Рисунок Г.18 — Зависимость показателей GCR и ICR от значения параметра β на контрольном множестве из 23214 элементов (6 детекторов 10–15 $\times$ 10–15)

Таблица 16 — Показатели эффективности классификаторов в схеме one-vs-all

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	РБФ	РНС	ММГ	МВГ	МФХ	ММУ
TPR	min	98.682	98.861	97.200	98.109	53.198	98.295	98.853	99.717	98.208
	max	99.836	99.784	99.784	98.995	77.050	98.660	99.821	99.851	99.866
	avg	98.871	99.543	99.058	98.759	66.570	98.481	99.436	99.796	99.626
FPR	min	0.349	0.340	0.270	0.307	0.000	0.074	0.146	0.294	0.081
	max	1.338	1.820	1.389	1.323	88.391	0.866	0.868	0.589	1.489
	avg	0.448	0.710	0.646	0.824	13.228	0.330	0.467	0.374	0.677
CCR	min	97.533	97.080	98.275	97.260	34.926	97.794	97.882	99.290	98.338
	max	98.087	98.266	99.222	97.866	87.954	98.226	98.503	99.482	99.557
	avg	98.021	98.029	98.860	97.545	77.559	98.084	98.347	99.411	99.070
ICR	min	1.904	1.729	0.580	2.134	12.046	1.774	1.497	0.496	0.443
	max	2.467	2.892	1.680	2.740	65.074	2.206	2.118	0.659	1.662
	avg	1.976	1.961	0.970	2.452	22.439	1.916	1.653	0.562	0.930
GPR	min	99.313	99.423	99.558	99.448	73.211	99.166	99.411	99.669	99.669
	max	99.779	99.767	99.767	99.791	98.454	99.362	99.742	99.840	99.828
	avg	99.431	99.632	99.653	99.659	89.200	99.278	99.602	99.751	99.755
OPR	min	−1.991	−1.519	−6.372	−1.576	−70.240	−2.638	−1.507	0.003	−6.518
	max	0.140	0.279	0.119	0.122	−52.726	−1.743	0.246	0.279	0.225
	avg	−1.551	−0.224	−1.509	−0.324	−58.013	−2.189	−0.461	0.118	−0.587
GCR	min	99.010	98.466	98.809	98.738	21.126	99.334	99.213	99.571	99.001
	max	99.642	99.684	99.868	99.414	99.077	99.659	99.653	99.862	99.820
	avg	99.556	99.381	99.449	99.166	85.827	99.594	99.528	99.789	99.525
OCR	min	−7.408	−7.064	−6.165	−6.802	−65.788	−7.379	−6.446	−1.346	−6.425
	max	−6.491	−4.704	−0.939	−5.883	6.768	−6.781	−4.496	−0.745	−0.108
	avg	−7.146	−6.377	−2.795	−6.344	−51.008	−7.123	−5.540	−1.087	−2.006

Таблица 17 — Показатели эффективности классификаторов в схеме one-vs-one

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	РБФ	РНС	ММГ	МВГ	МФХ	ММУ
TPR	min	99.829	99.791	99.769	98.302	87.361	99.464	99.779	99.791	99.437
	max	99.844	99.851	99.836	99.851	99.841	99.846	99.846	99.873	99.903
	avg	99.838	99.826	99.792	99.467	94.602	99.780	99.821	99.844	99.769
FPR	min	0.352	0.281	0.273	0.330	2.162	0.314	0.314	0.301	0.330
	max	0.352	2.471	0.802	1.792	98.468	0.572	0.663	2.412	15.480
	avg	0.352	1.030	0.458	0.692	32.990	0.368	0.377	0.997	3.060
CCR	min	98.073	97.321	98.968	97.317	33.710	98.164	98.165	98.428	90.263
	max	98.542	98.846	99.661	98.048	89.995	99.026	99.026	99.697	99.508
	avg	98.367	98.269	99.295	97.711	71.878	98.485	98.516	99.272	97.712
ICR	min	1.458	1.127	0.339	1.952	6.990	0.974	0.974	0.303	0.492
	max	1.927	2.516	1.029	2.680	65.521	1.835	1.835	1.503	9.737
	avg	1.633	1.533	0.673	2.221	26.329	1.495	1.484	0.627	2.288
GPR	min	99.755	99.718	99.681	98.785	96.736	99.718	99.718	99.705	99.693
	max	99.804	99.791	99.804	99.804	100.000	99.791	99.804	99.816	99.877
	avg	99.779	99.759	99.725	99.638	98.759	99.746	99.756	99.774	99.786
OPR	min	0.055	0.092	0.055	−0.032	−25.735	−0.697	0.055	0.083	−0.722
	max	0.225	0.282	0.259	0.772	0.040	0.238	0.238	0.282	0.258
	avg	0.144	0.178	0.196	0.177	−7.852	0.087	0.166	0.181	0.044
GCR	min	99.594	99.562	99.560	98.633	17.395	99.592	99.592	99.724	92.941
	max	99.705	99.793	99.906	99.602	99.073	99.745	99.761	99.893	99.841
	avg	99.656	99.690	99.835	99.341	72.967	99.664	99.667	99.810	98.154
OCR	min	−7.362	−5.213	−3.835	−7.442	−47.999	−6.783	−6.768	−0.254	−1.938
	max	−5.127	−3.756	−0.392	−5.896	35.422	−2.904	−2.871	−0.019	3.704
	avg	−5.944	−4.440	−1.674	−6.768	−16.556	−5.283	−5.083	−0.139	−0.182

Таблица 18 — Показатели эффективности классификаторов в схеме CBT

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	РБФ	РНС	ММГ	МВГ	МФХ	ММУ
TPR	min	99.777	99.739	98.861	98.883	33.770	98.250	99.419	99.571	96.858
	max	99.844	99.844	99.762	99.844	93.919	99.727	99.829	99.955	99.873
	avg	99.826	99.793	99.619	99.540	76.602	99.062	99.755	99.840	99.409
FPR	min	0.298	0.276	0.276	0.350	0.253	0.061	0.273	0.263	0.352
	max	0.524	3.150	2.287	2.652	93.815	0.353	0.636	2.621	5.779
	avg	0.378	0.764	0.611	1.265	34.231	0.298	0.356	0.558	2.373
CCR	min	97.976	96.933	98.197	96.803	33.349	98.031	98.388	97.266	95.121
	max	98.085	98.638	99.351	98.428	88.080	98.648	98.712	98.655	99.378
	avg	98.058	98.319	98.918	97.743	65.069	98.363	98.591	98.439	97.835
ICR	min	1.915	1.362	0.649	1.572	11.920	1.352	1.288	1.345	0.622
	max	2.024	3.067	1.803	3.197	66.651	1.969	1.612	2.734	4.879
	avg	1.942	1.681	1.082	2.257	34.931	1.637	1.409	1.561	2.165
GPR	min	99.693	99.644	99.350	99.632	25.500	99.264	99.644	99.705	99.067
	max	99.804	99.791	99.705	99.816	99.840	99.779	99.791	99.890	99.828
	avg	99.767	99.732	99.620	99.739	83.556	99.475	99.740	99.799	99.682
OPR	min	0.075	0.092	0.111	0.037	−30.928	−2.264	−0.033	−0.158	−7.028
	max	0.246	0.275	0.449	0.210	29.921	0.122	0.275	0.262	0.189
	avg	0.153	0.175	0.225	0.104	−14.935	−1.067	0.145	0.121	−0.853
GCR	min	99.569	99.462	99.510	98.271	19.985	99.594	99.673	99.546	96.701
	max	99.682	99.780	99.851	99.629	96.125	99.776	99.778	99.799	99.807
	avg	99.612	99.710	99.741	99.039	68.973	99.664	99.703	99.728	98.794
OCR	min	−7.448	−5.644	−5.101	−6.523	−49.354	−6.838	−5.728	−6.328	−9.282
	max	−6.999	−4.662	−1.282	−3.503	16.899	−4.795	−4.475	−4.578	0.750
	avg	−7.200	−5.024	−2.598	−5.198	−26.326	−5.907	−4.793	−5.124	−2.707

Таблица 19 — Показатели эффективности классификаторов в схеме DAG

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	РБФ	РНС	ММГ	МВГ	МФХ	ММУ
TPR	min	99.829	99.791	99.762	98.935	71.048	97.677	99.784	98.801	97.277
	max	99.844	99.873	99.844	99.866	96.383	99.791	99.846	99.933	99.859
	avg	99.838	99.837	99.795	99.424	87.499	99.207	99.821	99.681	99.203
FPR	min	0.352	0.321	0.273	0.311	5.951	0.293	0.293	0.289	0.115
	max	0.352	2.405	3.829	0.983	96.053	0.353	0.396	0.884	6.292
	avg	0.352	0.836	0.882	0.581	31.297	0.328	0.335	0.415	2.484
CCR	min	98.073	97.583	97.453	97.411	34.177	98.170	98.201	97.505	95.628
	max	98.599	98.811	99.622	98.102	86.692	98.870	99.037	98.673	99.295
	avg	98.319	98.411	99.101	97.820	71.239	98.513	98.753	97.926	97.819
ICR	min	1.401	1.189	0.378	1.898	13.308	1.130	0.963	1.327	0.705
	max	1.927	2.417	2.547	2.589	65.823	1.830	1.799	2.495	4.372
	avg	1.681	1.589	0.899	2.180	28.761	1.487	1.247	2.074	2.181
GPR	min	99.755	99.730	99.681	99.067	94.601	98.883	99.705	99.705	99.276
	max	99.804	99.828	99.779	99.840	99.742	99.767	99.791	99.902	99.816
	avg	99.782	99.773	99.732	99.683	97.482	99.557	99.756	99.791	99.691
OPR	min	0.075	0.092	0.060	−0.061	−60.905	−3.357	0.072	−2.988	−5.111
	max	0.225	0.270	0.266	0.751	−5.962	0.084	0.238	0.295	0.181
	avg	0.140	0.167	0.189	0.143	−24.429	−0.762	0.170	−0.406	−0.805
GCR	min	99.594	99.529	98.972	99.115	18.095	99.611	99.638	99.483	97.258
	max	99.694	99.774	99.906	99.636	97.086	99.770	99.793	99.791	99.772
	avg	99.651	99.697	99.728	99.414	74.674	99.685	99.716	99.640	98.775
OCR	min	−7.418	−4.923	−4.051	−6.937	−58.922	−6.859	−6.682	−10.410	−7.387
	max	−4.858	−3.773	0.139	−6.255	28.534	−3.651	−3.036	−5.041	0.167
	avg	−6.167	−4.274	−1.295	−6.630	−26.581	−5.181	−4.272	−7.882	−1.640

Таблица 20 — Показатели эффективности классификаторов в схеме one-vs-all*

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	СК	ЛК	ММГ	МВГ	МФХ	ММУ
TPR	min	98.682	98.861	97.200	85.705	84.255	96.517	99.092	99.717	26.803
	max	99.836	99.784	99.784	85.978	84.744	98.047	99.779	99.844	98.898
	avg	98.871	99.543	99.058	85.773	84.622	97.555	99.548	99.796	58.029
FPR	min	0.349	0.340	0.270	85.797	100.000	0.186	0.294	0.294	23.534
	max	1.338	1.820	1.389	85.932	100.000	1.026	1.307	1.940	86.070
	avg	0.448	0.710	0.646	85.898	100.000	0.417	0.695	0.509	50.008
CCR	min	97.533	97.080	98.275	9.389	0.000	97.579	97.869	98.397	25.180
	max	98.087	98.266	99.222	9.515	0.000	98.177	98.500	99.414	67.289
	avg	98.021	98.029	98.860	9.420	0.000	97.974	98.336	99.266	44.739
ICR	min	1.904	1.729	0.580	18.835	100.000	1.808	1.500	0.571	32.711
	max	2.467	2.892	1.680	18.957	100.000	2.395	2.131	0.817	74.820
	avg	1.976	1.961	0.970	18.927	100.000	2.010	1.664	0.635	55.261
GPR	min	99.313	99.423	99.558	85.667	85.777	99.067	99.571	99.669	12.505
	max	99.779	99.767	99.767	85.986	86.011	99.337	99.755	99.840	99.914
	avg	99.431	99.632	99.653	85.747	85.836	99.175	99.686	99.751	51.689
OPR	min	−1.991	−1.519	−6.372	−0.762	−2.807	−6.759	−1.135	0.003	−27.611
	max	0.140	0.279	0.119	0.390	−2.697	−2.715	0.158	0.279	42.876
	avg	−1.551	−0.224	−1.509	−0.474	−2.779	−4.389	−0.363	0.120	14.912
GCR	min	99.010	98.466	98.809	12.030	0.000	99.129	98.905	98.765	23.975
	max	99.642	99.684	99.868	12.267	0.000	99.707	99.740	99.862	65.850
	avg	99.556	99.381	99.449	12.208	0.000	99.555	99.426	99.708	46.982
OCR	min	−7.408	−7.064	−6.165	−7.856	0.000	−9.827	−5.760	−1.757	−24.102
	max	−6.491	−4.704	−0.939	−7.468	0.000	−6.079	−4.222	−0.459	13.030
	avg	−7.146	−6.377	−2.795	−7.759	0.000	−7.621	−5.000	−1.375	−3.742

Таблица 21 — Показатели эффективности классификаторов в схеме one-vs-one*

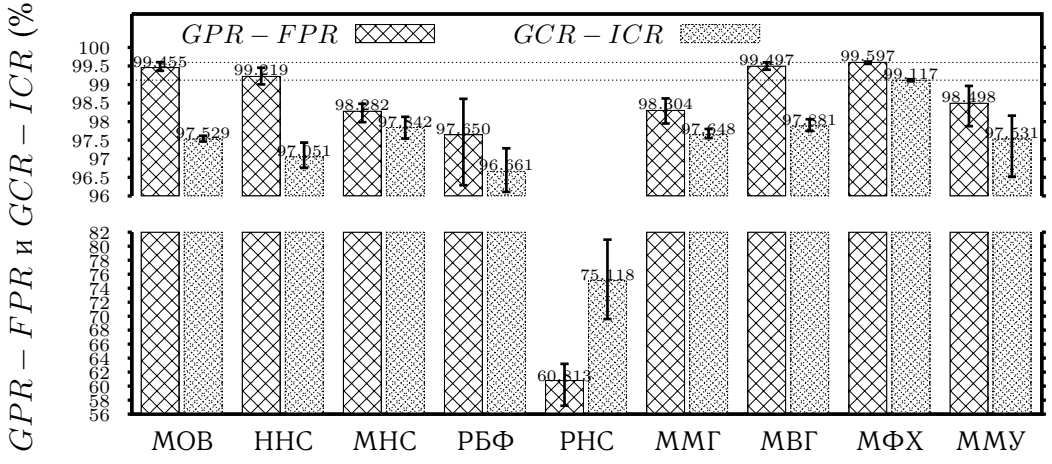
Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	СК	ЛК	ММГ	МВГ	МФХ	ММУ
TPR	min	99.829	99.791	99.769	85.705	84.255	96.957	99.782	99.791	21.600
	max	99.844	99.851	99.836	85.978	84.744	98.369	99.851	99.851	84.407
	avg	99.838	99.826	99.792	85.773	84.622	97.781	99.827	99.839	48.962
FPR	min	0.352	0.281	0.273	85.797	100.000	0.275	0.283	0.301	17.879
	max	0.352	2.471	0.802	85.932	100.000	0.840	1.116	2.443	81.265
	avg	0.352	1.030	0.458	85.898	100.000	0.416	0.517	0.995	43.263
CCR	min	98.073	97.321	98.968	9.389	0.000	98.051	98.471	98.403	20.885
	max	98.542	98.846	99.661	9.515	0.000	98.647	99.023	99.694	65.257
	avg	98.367	98.269	99.295	9.420	0.000	98.340	98.886	99.263	46.764
ICR	min	1.458	1.127	0.339	18.835	100.000	1.330	0.977	0.306	34.743
	max	1.927	2.516	1.029	18.957	100.000	1.873	1.529	1.478	79.115
	avg	1.633	1.533	0.673	18.927	100.000	1.612	1.114	0.633	53.236
GPR	min	99.755	99.718	99.681	85.667	85.777	99.141	99.718	99.705	13.486
	max	99.804	99.791	99.804	85.986	86.011	99.460	99.804	99.816	84.820
	avg	99.779	99.759	99.725	85.747	85.836	99.308	99.761	99.769	41.755
OPR	min	0.055	0.092	0.055	−0.762	−2.807	−5.925	0.055	0.083	−21.297
	max	0.225	0.282	0.259	0.390	−2.697	−2.905	0.270	0.282	38.912
	avg	0.144	0.178	0.196	−0.474	−2.779	−4.171	0.173	0.180	17.109
GCR	min	99.594	99.562	99.560	12.030	0.000	99.661	99.621	99.705	16.876
	max	99.705	99.793	99.906	12.267	0.000	99.768	99.814	99.893	72.498
	avg	99.656	99.690	99.835	12.208	0.000	99.731	99.737	99.806	51.154
OCR	min	−7.362	−5.213	−3.835	−7.856	0.000	−8.119	−3.875	−0.384	−30.566
	max	−5.127	−3.756	−0.392	−7.468	0.000	−4.867	−2.770	−0.007	18.816
	avg	−5.944	−4.440	−1.674	−7.759	0.000	−6.219	−3.206	−0.178	−6.139

Таблица 22 — Показатели эффективности классификаторов в схеме CBT*

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	СК	ЛК	ММГ	МВГ	МФХ	ММУ
TPR	min	99.777	99.739	98.861	85.705	84.255	96.788	99.695	94.562	17.182
	max	99.844	99.844	99.762	85.978	84.744	98.096	99.826	94.629	86.574
	avg	99.826	99.793	99.619	85.773	84.622	97.634	99.768	94.603	55.079
FPR	min	0.298	0.276	0.276	85.797	100.000	0.173	0.276	0.284	18.497
	max	0.524	3.150	2.287	85.932	100.000	0.732	1.785	2.208	67.866
	avg	0.378	0.764	0.611	85.898	100.000	0.344	0.709	0.491	39.907
CCR	min	97.976	96.933	98.197	9.389	0.000	97.546	97.807	87.593	25.776
	max	98.085	98.638	99.351	9.515	0.000	98.324	98.687	88.747	65.066
	avg	98.058	98.319	98.918	9.420	0.000	98.066	98.416	88.620	48.424
ICR	min	1.915	1.362	0.649	18.835	100.000	1.641	1.313	11.253	34.934
	max	2.024	3.067	1.803	18.957	100.000	2.394	2.193	12.380	74.224
	avg	1.942	1.681	1.082	18.927	100.000	1.895	1.584	11.375	51.576
GPR	min	99.693	99.644	99.350	85.667	85.777	99.031	99.644	99.166	13.732
	max	99.804	99.791	99.705	85.986	86.011	99.276	99.767	99.276	94.969
	avg	99.767	99.732	99.620	85.747	85.836	99.164	99.721	99.218	52.059
OPR	min	0.075	0.092	0.111	−0.762	−2.807	−6.047	0.096	−11.066	−23.136
	max	0.246	0.275	0.449	0.390	−2.697	−2.830	0.295	−10.854	32.470
	avg	0.153	0.175	0.225	−0.474	−2.779	−4.029	0.180	−10.966	6.754
GCR	min	99.569	99.462	99.510	12.030	0.000	99.640	99.556	99.207	28.300
	max	99.682	99.780	99.851	12.267	0.000	99.734	99.778	99.320	70.526
	avg	99.612	99.710	99.741	12.208	0.000	99.684	99.674	99.296	54.227
OCR	min	−7.448	−5.644	−5.101	−7.856	0.000	−9.026	−4.982	−52.305	−43.807
	max	−6.999	−4.662	−1.282	−7.468	0.000	−6.517	−4.483	−52.142	7.294
	avg	−7.200	−5.024	−2.598	−7.759	0.000	−7.366	−4.734	−52.204	−11.933

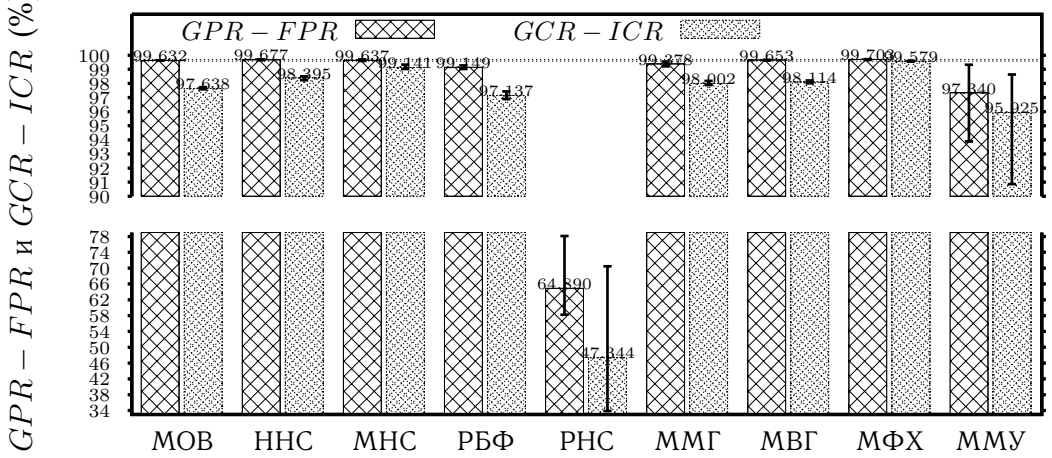
Таблица 23 — Показатели эффективности классификаторов в схеме DAG*

Показатель		Классификаторы и их агрегирующие композиции								
		МОВ	ННС	МНС	СК	ЛК	ММГ	МВГ	МФХ	ММУ
TPR	min	99.829	99.791	99.762	85.705	84.255	97.205	99.791	96.418	19.637
	max	99.844	99.873	99.844	85.978	84.744	98.483	99.851	97.150	92.990
	avg	99.838	99.837	99.795	85.773	84.622	97.685	99.834	96.561	43.021
FPR	min	0.352	0.321	0.273	85.797	100.000	0.214	0.311	0.289	7.898
	max	0.352	2.405	3.829	85.932	100.000	0.743	2.150	0.340	61.646
	avg	0.352	0.836	0.882	85.898	100.000	0.413	0.692	0.314	31.581
CCR	min	98.073	97.583	97.453	9.389	0.000	98.034	98.004	93.320	34.161
	max	98.599	98.811	99.622	9.515	0.000	98.646	99.088	94.545	62.828
	avg	98.319	98.411	99.101	9.420	0.000	98.304	98.799	93.528	52.411
ICR	min	1.401	1.189	0.378	18.835	100.000	1.336	0.912	5.455	37.172
	max	1.927	2.417	2.547	18.957	100.000	1.914	1.996	6.672	65.839
	avg	1.681	1.589	0.899	18.927	100.000	1.648	1.201	6.466	47.589
GPR	min	99.755	99.730	99.681	85.667	85.777	99.141	99.755	99.460	17.425
	max	99.804	99.828	99.779	85.986	86.011	99.423	99.791	99.705	97.362
	avg	99.782	99.773	99.732	85.747	85.836	99.267	99.772	99.535	38.057
OPR	min	0.075	0.092	0.060	−0.762	−2.807	−5.682	0.060	−8.290	−10.421
	max	0.225	0.270	0.266	0.390	−2.697	−2.486	0.245	−6.658	35.736
	avg	0.140	0.167	0.189	−0.474	−2.779	−4.163	0.162	−7.824	11.467
GCR	min	99.594	99.529	98.972	12.030	0.000	99.575	99.242	99.573	33.430
	max	99.694	99.774	99.906	12.267	0.000	99.772	99.805	99.745	79.984
	avg	99.651	99.697	99.728	12.208	0.000	99.712	99.691	99.633	60.425
OCR	min	−7.418	−4.923	−4.051	−7.856	0.000	−7.554	−3.533	−33.911	−50.950
	max	−4.858	−3.773	0.139	−7.468	0.000	−4.777	−2.546	−27.873	16.804
	avg	−6.167	−4.274	−1.295	−7.759	0.000	−6.232	−3.111	−32.645	−16.480



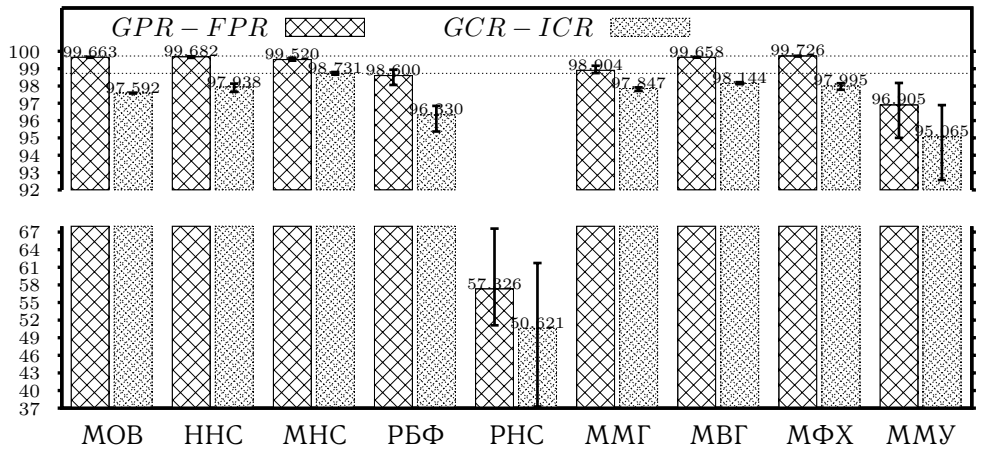
– $GPR \Uparrow$ на 0.021% (МФХ vs ННС)
– $FPR \Downarrow$ на 0.072% (ММГ vs МОВ)
– $GCR \Uparrow$ на 0.707% (МФХ vs МНС)
– $ICR \Downarrow$ на 0.569% (МФХ vs МНС)
– $GPR - FPR \Uparrow$ на 0.142% (МФХ vs МОВ)
– $GCR - ICR \Uparrow$ на 1.276% (МФХ vs МНС)

Рисунок Г.19 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема one-vs-all)



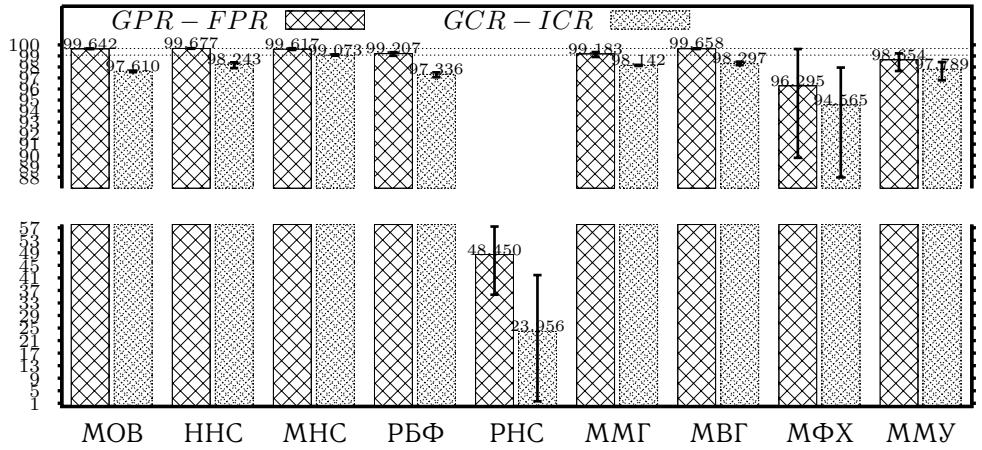
– $GPR \Uparrow$ на 0.003% (МФХ vs ННС)
– $FPR \Downarrow$ на 0.023% (МФХ vs ННС)
– $GCR \Uparrow$ на 0.217% (МФХ vs МНС)
– $ICR \Downarrow$ на 0.221% (МФХ vs МНС)
– $GPR - FPR \Uparrow$ на 0.025% (МФХ vs ННС)
– $GCR - ICR \Uparrow$ на 0.438% (МФХ vs МНС)

Рисунок Г.20 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема one-vs-one)



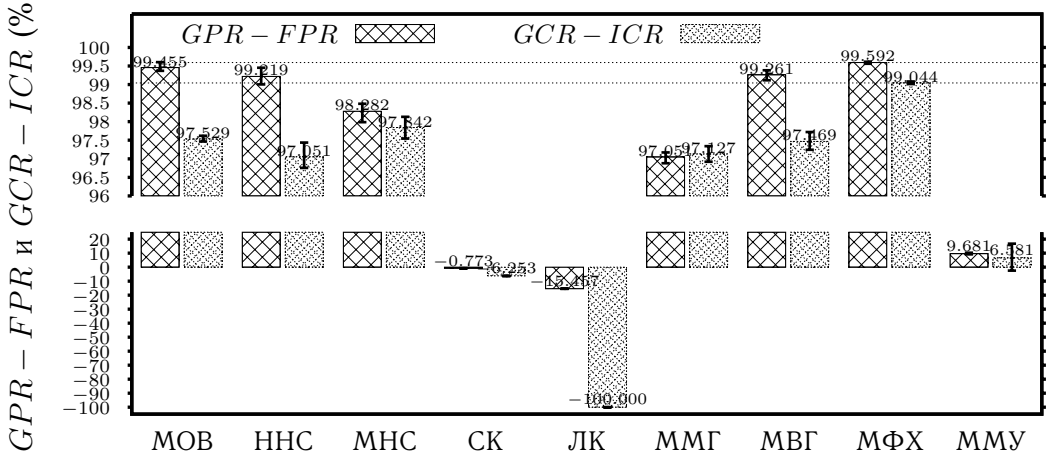
– $GPR \Uparrow$ на 0.011% (МФХ vs ННС)
– $FPR \Downarrow$ на 0.032% (МФХ vs ННС)
– $GCR \Downarrow$ на 0.294% (МВГ vs МНС)
– $ICR \Uparrow$ на 0.294% (МВГ vs МНС)
– $GPR - FPR \Uparrow$ на 0.043% (МФХ vs ННС)
– $GCR - ICR \Downarrow$ на 0.587% (МВГ vs МНС)

Рисунок Г.21 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема CBT)



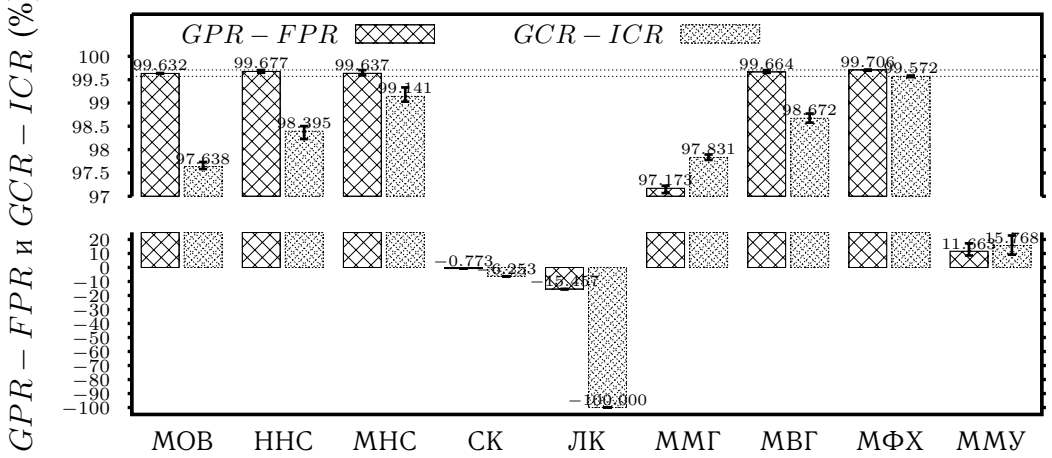
– $GPR \Downarrow$ на 0.025% (МВГ vs ННС)
– $FPR \Downarrow$ на 0.058% (МФХ vs ННС)
– $GCR \Downarrow$ на 0.388% (МВГ vs МНС)
– $ICR \Uparrow$ на 0.388% (МВГ vs МНС)
– $GPR - FPR \Downarrow$ на 0.020% (МВГ vs ННС)
– $GCR - ICR \Downarrow$ на 0.776% (МВГ vs МНС)

Рисунок Г.22 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема DAG)



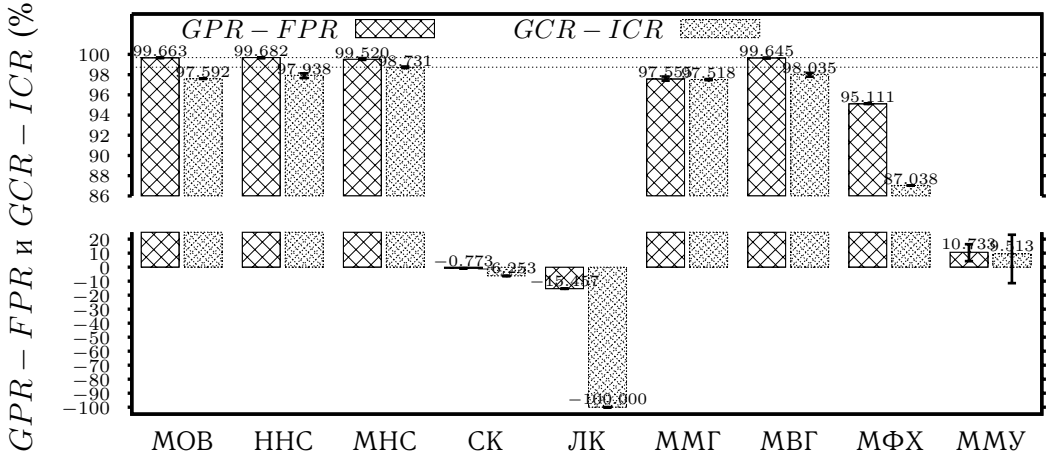
– $GPR \Uparrow$ на 0.017% (МФХ vs ННС)
– $FPR \Downarrow$ на 0.005% (МФХ vs МОВ)
– $GCR \Uparrow$ на 0.67% (МФХ vs МНС)
– $ICR \Downarrow$ на 0.533% (МФХ vs МНС)
– $GPR - FPR \Uparrow$ на 0.137% (МФХ vs МОВ)
– $GCR - ICR \Uparrow$ на 1.203% (МФХ vs МНС)

Рисунок Г.23 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема one-vs-all*)



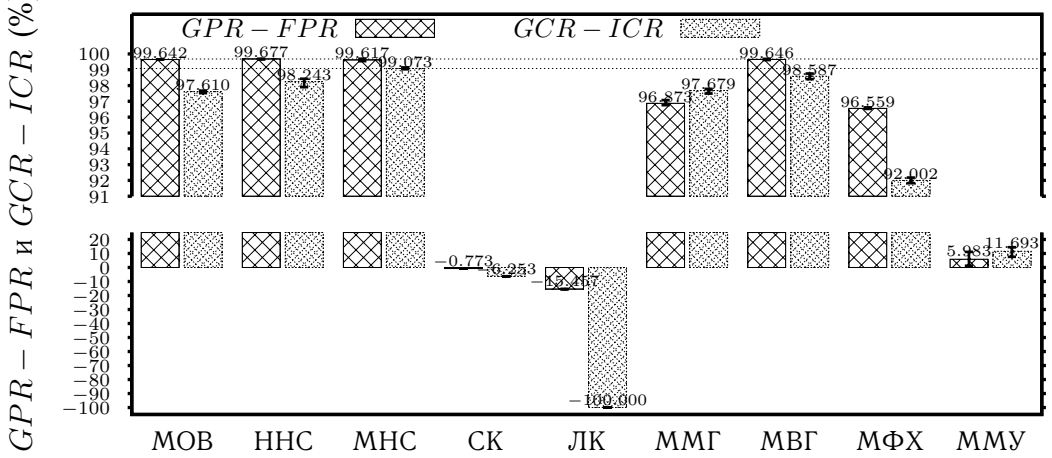
– $GPR \Uparrow$ на 0.003% (МФХ vs ННС)
– $FPR \Downarrow$ на 0.056% (ММГ vs ННС)
– $GCR \Uparrow$ на 0.213% (МФХ vs МНС)
– $ICR \Downarrow$ на 0.218% (МФХ vs МНС)
– $GPR - FPR \Uparrow$ на 0.029% (МФХ vs ННС)
– $GCR - ICR \Uparrow$ на 0.431% (МФХ vs МНС)

Рисунок Г.24 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема one-vs-one*)



– $GPR \Downarrow$ на 0.016% (МВГ vs ННС)
– $FPR \Downarrow$ на 0.035% (ММГ vs ННС)
– $GCR \Downarrow$ на 0.348% (МВГ vs МНС)
– $ICR \Uparrow$ на 0.348% (МВГ vs МНС)
– $GPR - FPR \Downarrow$ на 0.037% (МВГ vs ННС)
– $GCR - ICR \Downarrow$ на 0.696% (МВГ vs МНС)

Рисунок Г.25 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема CBT*)



– $GPR \Downarrow$ на 0.006% (МВГ vs ННС)
– $FPR \Downarrow$ на 0.05% (ММГ vs ННС)
– $GCR \Downarrow$ на 0.243% (МВГ vs МНС)
– $ICR \Uparrow$ на 0.243% (МВГ vs МНС)
– $GPR - FPR \Downarrow$ на 0.031% (МВГ vs ННС)
– $GCR - ICR \Downarrow$ на 0.487% (МВГ vs МНС)

Рисунок Г.26 — Показатели эффективности $GPR - FPR$, $GCR - ICR$, вычисленные с помощью пятиблочной кроссвалидации для пяти базовых классификаторов и четырех композиций (схема DAG*)

Таблица 24 — Временные и системные затраты обучения и тестирования коллектива классификаторов

Индикаторы			Схемы комбинирования			
			one-vs-all	one-vs-one	CBT	DAG
Обучение: количество обучающих экземпляров — 7000						
однопоточный режим	Время (сек.)	min	11 235.000	4569.000	4679.000	4660.000
		max	12 228.000	5218.000	5092.000	6148.000
		avg	11 822.000	4967.333	4880.000	5217.333
	Загрузка CPU (%)	min	92.700	92.700	92.700	92.700
		max	100.000	100.000	100.000	100.000
		avg	99.215	99.194	99.219	99.220
	Потребление RAM (MB)	min	62.496	64.309	62.492	64.324
		max	132.352	98.996	134.219	99.563
		avg	105.487	97.688	100.067	97.435
многопоточный режим	Время (сек.)	min	1782.000	754.000	900.000	743.000
		max	6996.000	875.000	1228.000	880.000
		avg	2767.429	823.429	1059.143	831.571
	Загрузка CPU (%)	min	92.700	92.700	92.700	92.800
		max	800.000	800.000	800.000	800.000
		avg	455.184	579.644	445.293	610.500
	Потребление RAM (MB)	min	62.496	64.305	62.504	64.324
		max	473.867	506.070	308.820	506.090
		avg	263.452	480.843	248.808	480.758
Тестирование: количество тестовых экземпляров — 101113						
однопоточный режим	Время (сек.)	min	383.000	985.000	176.000	296.000
		max	387.000	1019.000	182.000	301.000
		avg	384.667	996.667	179.000	298.000
	Загрузка CPU (%)	min	92.800	92.800	92.700	92.800
		max	100.000	100.000	100.000	100.000
		avg	99.197	99.177	99.123	99.212
	Потребление RAM (MB)	min	63.656	65.289	63.395	65.348
		max	88.957	90.816	88.125	90.500
		avg	82.348	84.368	79.987	83.629
многопоточный режим	Время (сек.)	min	644.000	1764.000	288.000	563.000
		max	701.000	1789.000	324.000	576.000
		avg	660.857	1776.857	308.286	569.429
	Загрузка CPU (%)	min	86.400	86.300	59.800	59.700
		max	153.000	186.100	113.000	172.800
		avg	113.028	119.584	99.806	100.081
	Потребление RAM (MB)	min	64.477	66.301	63.934	67.000
		max	91.031	92.215	88.473	91.250
		avg	82.987	84.877	80.397	84.525

Таблица 25 — Сравнение характеристик производительности COA

Характеристика		COA				
		netpcap_sensor	Snort	Suricata (single)	Suricata (autofp)	Bro
$T^{(real)}$ (sec.)	min	29.881	63.749	48.264	86.659	177.691
	max	31.094	72.340	60.049	120.151	195.778
	avg	30.657	65.682	53.121	102.743	184.730
$T^{(proc)}$ (sec.)	min	29.304	49.112	41.800	80.138	154.167
	max	30.512	55.975	53.569	113.668	175.034
	avg	30.071	51.340	46.618	96.192	160.801
$V^{(pkts)}\left(\frac{\text{Kpkts}}{\text{sec.}}\right)$	min	320.063	174.465	182.300	85.914	55.793
	max	333.257	198.844	233.627	121.860	63.345
	avg	324.756	190.215	209.483	101.522	60.731
$V^{(data)}\left(\frac{\text{Kbytes}}{\text{sec.}}\right)$	min	18 675.303	10 179.843	10 637.004	5012.965	3255.455
	max	19 445.207	11 602.359	13 631.906	7110.405	3696.090
	avg	18 949.140	11 098.823	12 223.090	5923.697	3543.591
$L^{(CPU)}$ (%)	min	52.100	0.000	0.000	0.000	6.600
	max	100.000	100.000	179.500	265.700	212.600
	avg	83.622	88.766	141.909	193.261	115.972
$C^{(virt)}$ (Mbytes)	min	176.098	77.785	0.000	0.000	0.000
	max	256.121	800.371	675.496	1026.121	1528.699
	avg	233.353	705.731	617.404	914.086	972.210
$C^{(resd)}$ (Mbytes)	min	144.262	14.664	0.000	0.000	0.000
	max	224.469	527.496	332.332	360.129	991.828
	avg	201.585	422.200	303.798	318.679	634.394
$C^{(real)}$ (Mbytes)	min	141.203	10.758	0.000	0.000	0.000
	max	221.234	522.137	325.188	352.902	982.750
	avg	198.452	416.870	296.685	311.556	625.444

Приложение Д

Копии актов о внедрении результатов диссертационной работы

Blekinge Institute of Technology (Karlskrona, Sweden)

Karlskrona, November 16, 2017

To whom it may concern:

Contribution of Alexander A. Branitskiy to the ENGENSEC (Educating the Next generation experts in Cyber Security) project

As a general manager of ENGENSEC project, I can declare that Mr. Alexander A. Branitskiy provided valuable contributions to this project in the following areas:

1. Design of the model of a computational immune system based on an evolutionary approach with an application to the detection of anomalous network connections;

2. Design of the algorithm of genetic-competitive learning of a Kohonen map with an application to the detection of anomalous network connections;

3. Design of the technique of hierarchical hybridization of binary classifiers with an application to the detection of anomalous network connections;

4. Development of architecture and software implementation of a distributed attack detection system based on combination of signature analysis and computational intelligence methods.

All the results are of equal importance to ENGENSEC project. These results were used while preparing the course "Advanced Network & Cloud Security" for master's training in the partner universities.

General manager of ENGENSEC project (Research Project of the European Community program TEMPUS № 544455-TEMPUS-1-2013-1-SE-TEMPUS-JPCR)

Anders Carlsson

«Educating the Next generation experts in Cyber Security (ENGENSEC)»

CERTIFICATE

PRESENTED TO

Alexander Branitskiy

Who has participated in the international
workshop in cyber security
in the ENGENSEC Tempus project.

Blekinge Institute of Technology
SE-371 79 Karlskrona
www.bth.se

Anders Carlsson
ENGENSEC GENERAL MANA

ENGENSEC

Co-funded by the
Tempus Programme
of the European Union

Технологический институт Блекинге (Карлскруна, Швеция)

Карлскруна, 16 ноября 2017 г.

В отношении кого рассматривается:

Вклад Браницкого Александра Александровича в проект ENGENSEC (Educating the Next generation experts in Cyber Security: the new EU-recognized Master's program)

Как генеральный менеджер проекта ENGENSEC, я могу заявить, что Браницкий Александр Александрович предоставил ценный вклад в этот проект в следующих областях:

1. Разработка модели вычислительной иммунной системы на базе эволюционного подхода с приложением к обнаружению аномальных сетевых соединений;

2. Разработка алгоритма генетико-конкурентного обучения сети Кохонена с приложением к обнаружению аномальных сетевых соединений;

3. Разработка методики иерархической гибридизации бинарных классификаторов с приложением к обнаружению аномальных сетевых соединений;

4. Разработка архитектуры и программная реализация распределенной системы обнаружения атак на основе комбинирования сигнатурного анализа и методов вычислительного интеллекта.

Все результаты являются одинаково важными для проекта ENGENSEC. Эти результаты использовались при подготовке курса "Современная сетевая и облачная безопасность" для обучения магистров в университетах-партнерах.

Генеральный менеджер проекта ENGENSEC (исследовательский проект программы TEMPUS Европейского сообщества № 544455-TEMPUS-1-2013-1-SE-TEMPUS-JPCR)

Андерс Карлссон

Правильность перевода и
его соответствие оригиналу
удостоверяю.

/И.П. подпись/

зам. директора по международ-
ным связям, СПИИРАН

Федеральное государственное бюджетное учреждение науки • Санкт-Петербургский институт информатики и автоматизации Российской академии наук

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)

«13» июня 2018 г. № 1

Санкт-Петербург

УТВЕРЖДАЮ
Проректор по научной работе
к.т.н., доцент
_____ Дукельский К.В.
«13» июня 2018 г.

Акт

об использовании результатов диссертационной работы
Браницкого Александра Александровича
«Обнаружение аномальных сетевых соединений на основе гибридизации методов вычислительного интеллекта» в учебном процессе университета

Настоящий Акт составлен в том, что результаты диссертационной работы Браницкого Александра Александровича, а именно:

- методика иерархической гибридизации бинарных классификаторов для обнаружения аномальных сетевых соединений;
- архитектура и программная реализация распределенной системы обнаружения атак, построенной на основе гибридизации методов вычислительного интеллекта и сигнатурного анализа.

используются кафедрой Защищенных систем связи СПбГУТ в учебном процессе на старших курсах обучения студентов по направлению подготовки бакалавров 10.03.01 «Информационная безопасность» по дисциплинам «Программно–аппаратные средства обеспечения информационной безопасности» и «Основы информационной безопасности» при чтении курсов лекций, проведении практических и лабораторных работ.

Заведующий кафедрой ЗСС,
к.т.н., доцент _____ Красов А.В.

Ученый секретарь кафедры ЗСС,
к.т.н., доцент _____ Кушнир Д.В.

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

УТВЕРЖДАЮ
Декан факультета безопасности информационных технологий
к.т.н., доцент
_____ Заколдаев Д. А.
«11» 06 2018 г.

Акт

об использовании результатов диссертационной работы
Браницкого Александра Александровича
«Обнаружение аномальных сетевых соединений на основе гибридизации методов вычислительного интеллекта» в учебном процессе университета

Настоящий Акт составлен в том, что результаты диссертационной работы Браницкого Александра Александровича, а именно:

- модель искусственной иммунной системы на базе эволюционного подхода для классификации сетевых соединений;
- алгоритм генетико-конкурентного обучения сети Кохонена для обнаружения аномальных сетевых соединений.

используются кафедрой ПБКС (проектирования и безопасности компьютерных систем) Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики в учебном процессе при подготовке магистров по направлению 10.04.01 «Информационная безопасность» по дисциплинам «Методология информационной безопасности» и «Основы форензики» при чтении курсов лекций, проведении практических и лабораторных работ.

Заведующий кафедрой ПБКС,
к.т.н., доцент Д.А. Заколдаев

Секретарь кафедры ПБКС Е.Н. Коваль