

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ГИБРИДНАЯ СИСТЕМА АНАЛИЗА
СЕТЕВОГО ТРАФИКА И УПРАВЛЕНИЯ
NFTABLES

Студент

30.05.25 Аверочкин Павел Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель

30.05.2025 Сорокин А.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

30.05.25 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

9 июня 2025 г. № 24)

Секретарь ГЭК

Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» 05 2025г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Аверочкин Павел Сергеевич гр. 21ВВП1 факультета вычислительной техники

2. Тема работы Гибридная система анализа трафика и управления nftables

Тема утверждена приказом ПГУ № 601/о от "6" 05 2025г.

3. Руководитель бакалаврской работы доцент кафедры ВТ Сорокин А.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

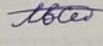
Разработать гибридную систему IDS/IPS, сочетающую сигнатурные и эвристические методы анализа сетевого трафика, с возможностью автоматического обнаружения и блокировки вредоносных IP-адресов в режиме реального времени. Языки программирования: C++, Python, Shell, CMake. Программа должна эффективно работать на ограниченных ресурсах Linux-сервера, иметь модульную структуру и использовать следующие технологии:

1. Docker
2. Conan
3. CMake
4. Многопоточность

5. Перечень подлежащих разработке вопросов

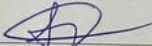
Требуется провести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки гибридной системы анализа трафика. Выбрать язык программирования и среду разработки.

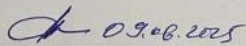
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	10%	06.05.2025-08.05.2025	
2	Анализ требований	10%	09.05.2025 – 13.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	5%	14.05.2025-16.05.2025	
4	Создание основной структуры системы, окружения.	20%	15.05.2025-20.05.2025	
5	Разработка и интеграция всех компонентов системы	25%	21.05.2025-24.05.2025	
6	Тестирование и отладка системы	10%	25.05.2025-26.05.2025	
7	Оформление пояснительной записки	20%	26.05.2025-30.05.2025	

Дата выдачи задания " 6 " 05 20 25 г.

Руководитель бакалаврской работы  Сорокин А.А.
(подпись, дата)

Задание к исполнению принял студент  Аверочкин П.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  09.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 154 листа, 37 рисунков, 3 таблицы, 11 источников, 3 приложения.

Цель работы – разработать гибридную систему анализа трафика и управления nftables.

Гибридная система должна работать в режиме реального времени, захватывать трафик на адаптере, анализировать его в двух плоскостях: сигнатурном и эвристическом. Результат анализа должен выводиться в журнал, у каждого вывода должен быть свой уровень важности. Конфигурирование программы должно производиться без пересборки из JSON-файла. Реализовать возможность включения белого списка для разрешенных известных IP-адресов.

В результате было разработано программа, выполняющая функции сервиса IPS/IDS, реализация удовлетворяет всем поставленным задачам. Используемые языки программирования: C++, Shell, Python, CMake. Система работает на ОС Linux и разрабатывалась в VSCode

ПГУ1.090301.1.001 ПЗ

Плм	Лист	№ докум	Подпись	Дата
Разработ		Аверочкин П.С.		20.05.20
Пров		Сорокин А.А.		20.05.20
Н. Контр.		Бычков А.С.		20.05.20
Утв		Митрохин М.А.		20.05.20

Гибридная система анализа
сетевого трафика и
управления nftables

Литера	Лист	Листов
	2	
ФВТ, каф. ВТ гр. 21ВВП1		

Содержание

Реферат	Error! Bookmark not defined.
Перечень сокращений и обозначений	5
Введение	6
1 Гибридная система IDS/IPS «HybridIDS» для ОС Linux.....	8
1.1 Анализ предметной области	8
1.2 Описание потребностей пользователей	9
1.3 Анализ аналогичных решений.....	10
1.3.1 Зарубежные решения	10
1.3.2 Отечественные решения	11
1.4 Проблематика	12
1.5 Вывод	12
2 Проектирование системы HibridIDS	13
2.1 Назначение и цели проекта	13
2.2 Взаимодействие компонентов	13
2.3 Конфигурация и форматы данных:	14
2.4 Средства сборки и развертывания.....	14
2.5 Логика обработки трафика	15
3 Настройка окружения и среды разработки	17
3.1 Выбор инструментов разработки	17
3.2 Обоснование выбора инструментов.....	17
3.2.1 Выбор языка программирования.....	17
3.2.2 Выбор окружения и набора инструментов	20
3.3 Установка и настройка виртуальной машины	22
3.4 Подключение к серверу по SSH	24
3.5 Установка и настройка инструментов в Ubuntu	26
3.6 Работа с удаленным репозиторием	26
4 Реализация системы	28
4.1 Структура проекта.....	28

4.2	Основной поток (main).....	29
4.3	Модуль работы с конфигурационным файлом (config_utils).....	31
4.4	Объектно-ориентированное представление сетевого пакета (packet)....	33
4.5	Компонент захвата трафика (sniffer)	34
4.6	Сигнатурный анализ (rule_engine).....	37
4.7	Эвристический анализ (heuristic_engine)	39
4.8	Управление сетевыми правилами (nftables_control).....	42
4.9	Журналирование (logger)	44
4.10	Автоматизированная сборка (build.sh)	47
5	Тестирование	50
5.1	Unit-тесты	50
5.1.1	Тестирование модуля конфигурации	50
5.1.2	Тестирование эвристического анализа.....	51
5.1.3	Тестирование сигнатурного анализа	52
5.1.4	Тестирование сниффера	52
5.1.5	Тестирование nftables-модуля	53
5.1.6	Тестирование модуля журналирования	53
5.1.7	Общие выводы и результаты	53
5.2	Практическое тестирование	55
5.2.1	Тестирование эвристики	55
5.2.2	Тестирование сигнатурного анализа	61
5.2.3	Вывод	67
6	Охрана труда.....	68
6.1	Электробезопасность	69
	Заключение	71
	Список использованных источников	75
	Приложение А	76
	Приложение Б.....	81
	Приложение В	147

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения.

ВМ – виртуальная машина

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

IDE – (от англ. Integrated Development Environment) – интеграционная среда разработки

ICMP – (от англ. Internet Control Message Protocol) – протокол управления сообщениями в Интернете

IDS – (от англ. Intrusion Detection System) – система обнаружения вторжений

IPS – (от англ. Intrusion Prevention System) – система предотвращения вторжений

JSON – (от англ. JavaScript Object Notation) – текстовый формат обмена данными

TCP – (от англ. Transmission Control Protocol) – протокол управления передачей

VB – Virtual Box

UDP – (от англ. User Datagram Protocol) – протокол пользовательских дейтаграмм

UML – (от англ. Unified Modeling Language) – унифицированный язык моделирования

Введение

Наступление XXI века ознаменовалось стремительным развитием цифровых технологий, проникновением интернета и сетевых решений практически во все сферы жизни: от бизнеса и государственной инфраструктуры до повседневного быта каждого человека. С каждым годом возрастает объем передаваемых данных, увеличивается число подключённых устройств, расширяется спектр онлайн-сервисов и приложений. Однако вместе с прогрессом в области информационных технологий стремительно растёт и количество угроз, сопряжённых с кибербезопасностью.

Одной из наиболее уязвимых составляющих цифровой инфраструктуры остаётся сеть, через которую злоумышленники могут получить несанкционированный доступ к данным, внедрить вредоносный код, нарушить работу сервисов или использовать ресурсы системы в своих целях. В ответ на эти вызовы существуют технологии сетевой безопасности, среди которых особое значение имеют системы обнаружения и предотвращения вторжений — IDS и IPS[1].

IDS-системы предназначены для мониторинга сетевого трафика и выявления подозрительной активности на основе сигнатур, аномалий или статистических закономерностей. В свою очередь, IPS-системы обладают дополнительной функцией – они способны автоматически реагировать на обнаруженные угрозы, блокируя вредоносный трафик для минимизации ущерба. В совокупности эти инструменты играют ключевую роль в построении многоуровневой защиты корпоративных и пользовательских сетей.

На современном рынке представлено множество коммерческих и открытых решений в области IDS/IPS, таких как Snort, Suricata, Zeek и другие. Каждая из этих систем обладает своими преимуществами и ограничениями, которые определяют эффективность их применения в различных сценариях. Однако, несмотря на наличие мощных и функционально насыщенных платформ, всё ещё

наблюдается острая потребность в легких, гибких и производительных инструментах, способных эффективно работать в условиях ограниченных ресурсов, обеспечивать удобную интеграцию с системными утилитами и при этом легко настраиваться под конкретные задачи.

Кроме того, многие существующие решения отличаются высокой степенью сложности развертывания и конфигурирования, что затрудняет их использование в небольших организациях или проектных группах, не располагающих специализированным персоналом. Актуальными остаются вопросы масштабируемости, расширяемости, поддержки современных стандартов трафика и удобства управления.

Учитывая вышеизложенное, возникает необходимость разработки гибридной IPS/IDS-системы, ориентированной на компактность, производительность, открытость архитектуры и тесную интеграцию с системными средствами управления доступом. Разработка такой системы на языке программирования C++ позволяет достичь высокой скорости обработки трафика, реализовать эффективное управление ресурсами и обеспечить прямое взаимодействие с низкоуровневыми компонентами операционной системы, включая iptables и другие модули ядра.

					ПГУ1.090301.1.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Гибридная система IDS/IPS «HybridIDS» для ОС Linux

1.1 Анализ предметной области

Системы обнаружения вторжений предназначены для выявления широкого спектра сетевых атак и аномального поведения, которые могут свидетельствовать о попытке проникновения или нарушения нормальной работы информационной системы. В данном подразделе рассмотрим основные типы угроз, на которые ориентированы современные IDS-системы.

Выделяют следующие типы угроз:

– сканирование портов. Целью является выявление открытых портов и структуры сети. Злоумышленник отправляет большое количество пакетов на разные порты или узлы целевой системы, пытаясь определить, какие сервисы активны, какова структура сети. Это может быть предварительным этапом перед основной атакой, так как таким способом можно выявить слабые места цели. Инструменты: nmap, masscan, zmap. Обнаружить вмешательство можно с помощью фиксации большого количества входящих запросов за короткий интервал времени или обнаружения характерных признаков по TCP или UDP пакетам.

– DoS и DDoS атаки. Злоумышленник пытается вывести из строя систему или ее узел путем перегрузки. Атака инициируется из одного или нескольких источников и направлена на перегрузку системы. ICMP flood, SYN flood, HTTP GET/POST flood – все это – примеры таких атак. Обнаружить атаку можно путем выявлением резкого роста однотипного трафика и поведенческим анализом.

– эксплуатация уязвимостей. Существует для получения несанкционированного доступа, выполнения произвольного кода, обход

					ПГУ1.090301.1.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

механизмов авторизации. Как пример: SQL-инъекции; командные инъекции; buffer overflow. Обнаруживаются с помощью DPI.

– аномальное поведение Аномалии выявляются зачастую только поведенческим и эвристическим анализом. Примерами могут быть: резкое увеличение объема исходящего трафика или подключение к ранее неизвестным или неиспользуемым портам. Обнаруживаются применением эвристических правил, вычисления статистических отклонений и машинного обучения[3].

Изучив все вышеперечисленное можно сделать вывод, что для обеспечения комплексной защиты одного только сигнатурного анализа недостаточно. Современные атаки часто хорошо маскируются, развиваются и меняются, поэтому могут быть не обнаружены по уже готовым шаблонам. Поэтому гибридный подход реализованный в HybridIDS позволяет:

- с одной стороны – быстро и качественно реагировать на известные атаки;
- с другой – обнаружить новые нестандартные угрозы и пресечь их.

1.2 Описание потребностей пользователей

Целевой аудиторией системы являются:

- системные администраторы и DevOps-инженеры: используют инструмент для защиты корпоративной и облачной инфраструктуры от вторжений.
- специалисты по информационной безопасности: используют IPS/IDS для мониторинга трафика, реагирования на инциденты;
- малый и средний бизнес: нуждаются в простом и эффективном решении, не требующем высоких затрат;
- разработчики и энтузиасты: ищут доступные инструменты для обучения, экспериментов и кастомизации.

Основные требования со стороны пользователей:

- простота настройки;
- высокая производительность;

- надежность;
- доступность;
- открытый исходный код.

1.3 Анализ аналогичных решений

1.3.1 Зарубежные решения

- Snort – одна из известных open-source IPS/IDS-систем[2].

Плюсы:

- 1) обширная база сигнатур;
- 2) ое механизмы обнаружения вторжений в реальном времени;
- 3) поддержка широкого спектра протоколов.

Минусы:

- 1) слабо выявляет ранее неизвестные угрозы;
 - 2) отсутствие встроенного эвристического анализа трафика;
 - 3) требовательность к ресурсам.
- Suricata – современное IPS/IDS-решение с поддержкой многопоточности и расширенного анализа[11].

Плюсы:

- 1) высокая производительность;
- 2) возможность анализа в реальном времени.

Минусы:

- 1) сложность настройки;
- 2) требовательность к ресурсам;

- Zeek – IDS фокусирующаяся на эвристическом и поведенческом анализе трафика.

Плюсы:

- 1) гибкость и расширяемость;
- 2) большие возможности в анализе;

3) поддержка анализа нестандартных протоколов;

Минусы:

- 1) отсутствие сигнатурной базы «из коробки»;
- 2) сложность настройки.

1.3.2 Отечественные решения

На российском рынке представлено несколько систем обнаружения вторжений, пользующихся той или иной популярностью. Важным фактором играет сертификация данных комплексов.

– СТАФОР;

Плюсы:

- 1) сертифицирована ФСТЭК России;
- 2) удобный графический интерфейс;
- 3) поддерживает сценарный анализ.

Минусы:

- 1) закрытая архитектура, сложность интеграции с другими системами;
- 2) закрытый исходный код.

– Континент IPS;

Плюсы:

- 1) встроенные механизмы DPI;
- 2) хорошая поддержка сигнатур и модуля анализа поведения.

Минусы:

- 1) высокая стоимость.

– Attack Killer (Сбер).

Плюсы:

- 1) использование машинного обучения;
- 2) облачная архитектура;
- 3) интеграция с SIEM.

Минусы:

- 1) продукт нацелен на крупный бизнес;
- 2) зависимость от облака.

1.4 Проблематика

На практике часто возникает ситуация, когда сигнатурные IDS хорошо обнаруживают известные угрозы, но бессильны против новых или измененных атак. А, в свою очередь, эвристические системы способны выявлять ранее неизвестные шаблоны поведения при атаках, но часто дают ложные срабатывания, обладают плохой поддержкой сигнатурного анализа и требуют значительных ресурсов на обучение и настройку.

Таким образом, между точностью и объемом охвата существует компромисс. Некоторые коммерческие системы сочетают в себе оба подхода, но open-source решения редко представляют из себя гибридную архитектуру.

1.5 Вывод

В связи с этим, на рынке сохраняется потребность в системе, которая бы совмещала достоинства сигнатурного и эвристического анализа, оставаясь при этом:

- достаточно легкой;
- простой в настройке;
- расширяемой;
- доступной.

Такая система способна:

- повысить число обнаружения атак;
- снизить количество ложных срабатываний;
- адаптироваться к новым типам угроз без ручной настройки.

Эти задачи призвана решить разработанная мной в рамках дипломного проекта HybridIDS – гибридная система обнаружения вторжений, сочетающая точность сигнатурных методов и гибкость и адаптивность эвристических.

2 Проектирование системы HibridIDS

2.1 Назначение и цели проекта

Система HibridIDS предназначена для защиты узла, в данном случае, Linux-сервера, при работе в сети в реальном времени с применением гибридного подхода, сочетающего сигнатурный и эвристический методы анализа. Цель проекта – обеспечить быструю и точную идентификацию потенциальных угроз, минимизируя количество ложных срабатываний и сохраняя высокую производительность даже при интенсивной сетевой нагрузке. Система ориентирована на работу в локальных и корпоративных сетях и способна адаптироваться к различным сценариям использования благодаря легкой расширяемости, модульной архитектуре и настраиваемым правилам.

2.2 Взаимодействие компонентов

– Sniffer захватывает сетевой трафик в реальном времени, структурирует его и передает в Rule Engine и Neuristic Engine.

– Rule Engine проверяет пакеты на соответствие правилам считанным Config Utils в базе и возвращает результат исследования программе и Logger.

– Neuristic Engine анализирует пакеты на предмет поведенческих аномалий и так же возвращает результат исследования программе и Logger.

– Nftables Control при необходимости блокирует вредоносный трафик, вызывая системные команды к nftables, при этом обращаясь к Logger и записывая информацию в лог.

Визуально взаимодействие компонентов изображено в UML-диаграммах в приложении А.

					ПГУ1.090301.1.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

2.3 Конфигурация и форматы данных:

Система использует формат JSON для правил и конфигурации работы: rules.json - содержит сигнатурные правила в виде условий (IP, порты, протокол и пр.).

Пример записи правила приведен ниже.

```
[
  {
    "id": 1,
    "description": "Блокировать все TCP пакеты с порта 80",
    "protocol": "TCP",
    "src_port": 80,
    "action": "block"
  },

```

config.json - включает параметры запуска: интерфейс для захвата, лог-файл, уровни тревог и пр.

Пример записи конфигурации следует.

```
{
  "interface": "enp0s3",
  "log_level": "info",
  "alert_cooldown": 10,
  "heuristicPacketThreshold": 100,
  "heuristicTimeWindowSeconds": 10
}
```

whitelist.txt – содержит белые адреса, которые проверяться не будут.

Пример списка белых IP-адресов ниже.

```
10.0.0.1
192.168.1.50
127.0.0.1
```

2.4 Средства сборки и развертывания

Для удобства сборки и развёртывания используется следующий стек:

- CMake — кроссплатформенная система сборки, упрощающая управление зависимостями и настройками компиляции;
- Conan — менеджер пакетов для C++, применяемый для установки внешних библиотек (например, рсар);

– Docker — предоставляет изолированную среду сборки. Это упрощает переносимость и тестирование;

– build.sh – bash-скрипт, позволяющий автоматизировать сборку и развертывание системы.

Docker-файл включает:

- установку библиотек, отсутствующих в Conan;
- установку зависимостей (Conan, CMake);
- создание контейнера для сборки;
- сборку проекта.

Conan-файл включает:

- подтягивание зависимостей.

CMake-файл включает:

- линковку библиотек и ресурсов проекта;
- конфигурацию сборки исполняемого файла;
- конфигурацию сборки тестов.

2.5 Логика обработки трафика

Полный процесс обработки пакетов включает несколько этапов:

- захват трафика. Sniffer подключается к заданному интерфейсу и в реальном времени захватывает пакеты;
 - сигнатурная проверка. Каждому пакету присваивается результат проверки на совпадение с правилами из rules.json. При совпадении может быть выполнено действие блокировки;
 - эвристический анализ. Пакеты без сигнатурных совпадений дополнительно анализируются на предмет частых повторов, необычных флагов TCP, аномального поведения и др;
 - логирование. Все предупреждения и действия заносятся в лог-файл в формате JSON, что облегчает дальнейший анализ[6];

– реакция. При необходимости, вызывается команда блокировки IP-адреса через nftables, обеспечивая активную защиту.

					ПГУ1.090301.1.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

3 Настройка окружения и среды разработки

3.1 Выбор инструментов разработки

Для разработки и тестирования сетевого проекта на C++ с использованием системных библиотек и сетевых интерфейсов была выбрана следующий набор инструментов, приведенный в таблице 1.

Таблица 1 – Набор инструментов для разработки

Инструмент	Назначение
VirtualBox	Эмуляция Linux-сервера в изолированной среде
Ubuntu Server 22.04	Легковесная ОС для разработки и тестирования
VSCode + Remote SSH	Редактирование кода из основной системы
Git	Управление версиями
CMake	Сборка
Conan	Управление зависимостями
nftables	Контроль сетевого трафика

3.2 Обоснование выбора инструментов

3.2.1 Выбор языка программирования

Разработка высокопроизводительной, гибкой и масштабируемой системы обнаружения и предотвращения вторжений (IPS/IDS) требует тщательного выбора инструментов, начиная с базового — языка программирования. От него зависит не только эффективность и скорость работы системы, но и удобство поддержки, возможность взаимодействия с сетевыми интерфейсами и

системными утилитами, а также степень контроля над ресурсами и безопасностью.

Критерии выбора:

- производительность;
- низкоуровневый доступ к сетевым и системным API;
- возможность интеграции с утилитами ядра Linux;
- наличие библиотек для сетевого анализа;
- поддержка многопоточности и параллелизма;
- кроссплатформенность.

На основе этих критериев были рассмотрены следующие языки: C, C++, Rust, Go, Python и Java. Достоинства и недостатки каждого приведены в таблице номер 2.

Таблица 2 – Достоинства языков программирования

Язык	Производительность	Безопасность памяти	Простота разработки	Поддержка сетевых библиотек	Доступ к системным вызовам
C	Очень высокая	Низкая	Средняя	Высокая	Прямой
C++	Очень высокая	Средняя	Средне-высокая	Очень высокая	Прямой
Rust	Высокая	Очень высокая	Низкая-средняя	Развивающаяся	Прямой
Go	Средне-высокая	Высокая	Высокая	Хорошая	Ограниченный
Python	Низкая	Средняя	Очень высокая	Очень высокая	Через обертки

Java	Средняя	Высокая	Средняя	Хорошая	Ограниченн ый
------	---------	---------	---------	---------	------------------

С учётом вышеизложенного, C++ является оптимальным языком программирования для реализации гибридной IPS/IDS-системы. Он предоставляет:

- высокую производительность, что критично при анализе трафика в реальном времени;
- низкоуровневый доступ к сетевым интерфейсам, системным вызовам и инструментам управления трафиком (включая nftables, netfilter и libpcap);
- гибкость архитектуры: C++ позволяет реализовывать как процедурные, так и объектно-ориентированные или даже шаблонные подходы, обеспечивая читаемость и масштабируемость кода;
- богатую экосистему библиотек: от Boost и POCO до современных инструментов анализа сетей, что облегчает реализацию функциональности фильтрации, логирования, работы с потоками и базами данных;
- многопоточность и асинхронность — возможность эффективно обрабатывать параллельные соединения и масштабировать систему под высокую нагрузку.

Также стоит отметить, что современные версии C++ (начиная с C++11) значительно упростили синтаксис, улучшили управление памятью (через smart pointers) и повысили безопасность разработки, приблизив удобство к языкам более высокого уровня, сохраняя при этом полный контроль над системой.

Альтернативы:

- Rust – отличный выбор с точки зрения безопасности и контроля, но требует большего времени на разработку;
- Go – подойдет для создания распределённых компонентов, но не для системного уровня управления трафиком;

– Python – прекрасен для анализа логов, написания сигнатур или администрирования, но не применим для основной высоконагруженной части системы.

3.2.2 Выбор окружения и набора инструментов

Окружение для разработки и работы программы были выбраны VirtualBox + Ubuntu Server. Почему не Windows: Проект ориентирован на сетевой анализ с использованием Linux-утилит и nftables, которые не поддерживаются в Windows. Почему не WSL: WSL не поддерживает полноценный доступ к сетевому стеку на уровне ядра (например, libpcap, raw-сокеты, iptables/nftables), что критично для IDS.

Преимущества Ubuntu Server:

- легковесность: минимальные системные требования;
- популярность: богатая документация, стабильные пакеты;
- полноценный доступ к сетевым интерфейсам и утилитам.

Средой разработки и способом подключения к виртуальной машине были выбраны Visual Studio Code + Remote SSH. Почему не работать напрямую в консоли: консоль хороша для сборки и запуска, но не обеспечивает удобную навигацию по коду, автодополнение, просмотр документации, отладку.

Преимущества VSCode:

- кроссплатформенность и лёгкость;
- расширение Remote - SSH позволяет работать с кодом на удалённой машине так, как будто он локальный;
- поддержка CMake, GDB, Git, автодополнение и шаблоны C++;
- встроенная поддержка .vscode/settings.json, которая уже настроена в проекте.

Инструментом для контроля версий и публикации на Github был выбран Git.

Преимущества Git:

- де-факто стандарт в индустрии для управления версиями;
- возможность отката, сравнения изменений;
- прямая интеграция с VSCode и GitHub;
- локальный и удалённый контроль.

Инструментами для сборки и подтягивания зависимостей были выбраны CMake + Conan.

Почему не Make или g++:

- проект состоит из нескольких модулей с заголовками, исходниками, конфигурациями;
- CMake упрощает переносимость и сборку как на Linux, так и на других системах[9].

Преимущества Conan:

- упрощает работу с зависимостями (например, библиотеками логирования, анализа пакетов и др.)[10];
- лучше, чем ручная установка зависимостей и прописывание путей вручную;

Инструментом для работы с сетевыми аспектами была выбрана библиотека nftables.

Почему nftables, а не iptables:

- nftables — современная система фильтрации и маршрутизации пакетов, официально заменяющая iptables;
- поддерживает более удобный синтаксис и имеет более высокую производительность;
- позволяет реализовывать фильтрацию и правила прямо из программы, что делает IDS более гибкой.

С данным стеком я работаю достаточно давно, поэтому были выбраны именно эти инструменты, а не их аналоги.

3.3 Установка и настройка виртуальной машины

С официального сайта <https://www.virtualbox.org/> скачана и установлена Oracle VirtualBox. Показано на рисунке 1.

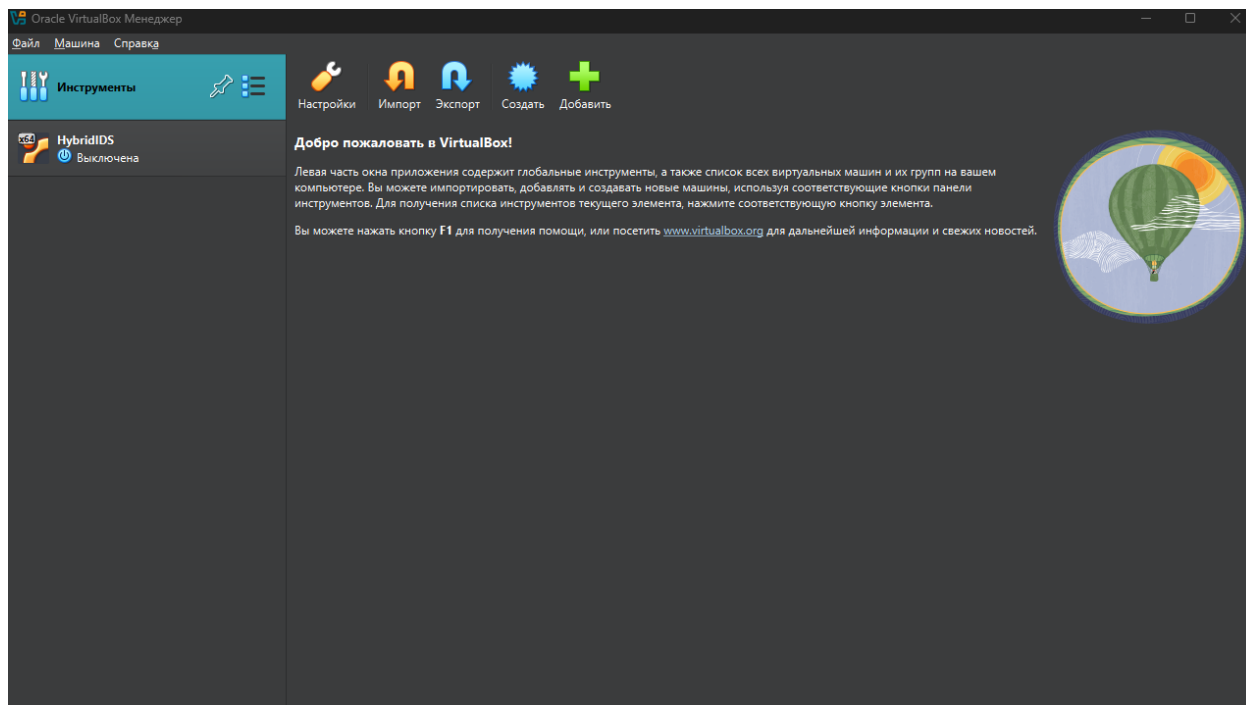


Рисунок 1 – Приветственное окно VB

С официального сайта <https://ubuntu.com/download/server> был скачан образ Ubuntu Server 22.04. Следом была создана виртуальная машина со следующими характеристиками: 4 CPU, 4096 RAM, 20 GB disk. Созданная VM изображена на рисунке 2.

					ПГУ1.090301.1.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

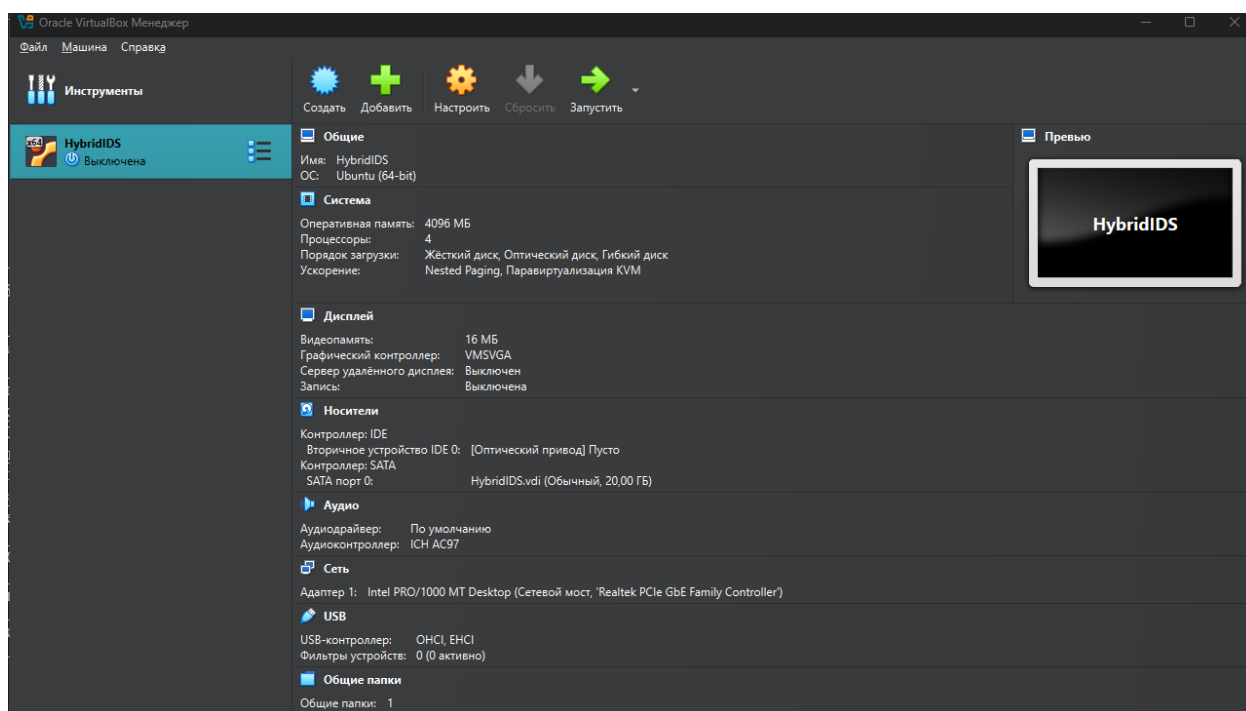


Рисунок 2 – Создание виртуальной машины

Для работы внутри локально сети через хост на одном из адаптеров был установлен режим работы: Сетевой мост. Показано на рисунке 3 и 4.

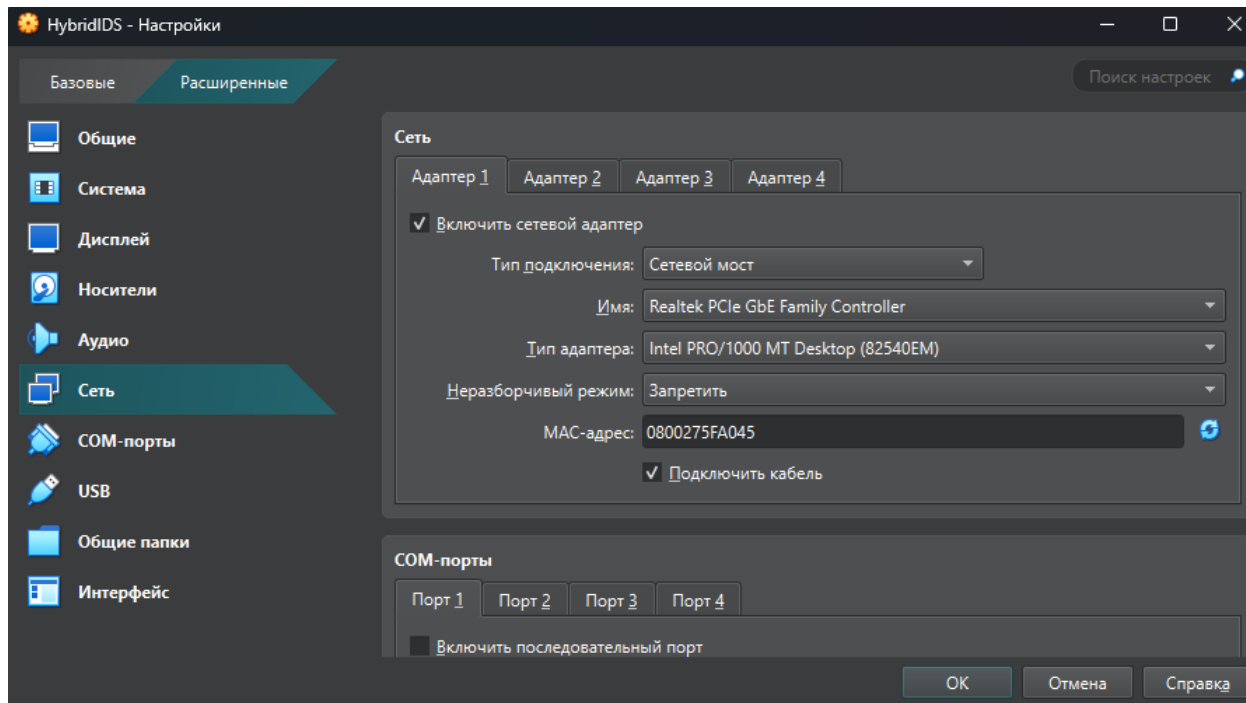


Рисунок 3 – Настройки виртуальной машины

После установки системы бал добавлен пользователь admin.

```
admin@HybridIDS:~$ whoami
admin
admin@HybridIDS:~$ _
```

Рисунок 4 – Пользователь

Чтобы была возможность подключения по SSH с хоста был установлен и запущен SSH Server. Можем наблюдать на рисунке 5.

```
admin@HybridIDS:~$ sudo systemctl -a | grep "ssh"
ssh.service                                loaded active running OpenBSD Secure
ssh.socket                                loaded active running OpenBSD Secure
admin@HybridIDS:~$ sudo systemctl status ssh.service
• ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; preset: enabled)
  Active: active (running) since Tue 2025-06-03 20:18:52 UTC; 2min 37s ago
  TriggeredBy: • ssh.socket
  Docs: man:sshd(8)
        man:sshd_config(5)
  Process: 927 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Main PID: 945 (sshd)
  Tasks: 1 (limit: 4605)
  Memory: 2.2M (peak: 2.4M)
  CPU: 28ms
  CGroup: /system.slice/ssh.service
          └─945 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Jun 03 20:18:52 HybridIDS systemd[1]: Starting ssh.service - OpenBSD Secure Shell server...
Jun 03 20:18:52 HybridIDS sshd[945]: Server listening on :: port 22.
Jun 03 20:18:52 HybridIDS systemd[1]: Started ssh.service - OpenBSD Secure Shell server.
admin@HybridIDS:~$ _
```

Рисунок 5 – Установка SSH-сервера

3.4 Подключение к серверу по SSH

На хост-машине с официального сайта был скачан и установлен VSCode. Изображено на рисунке 6.

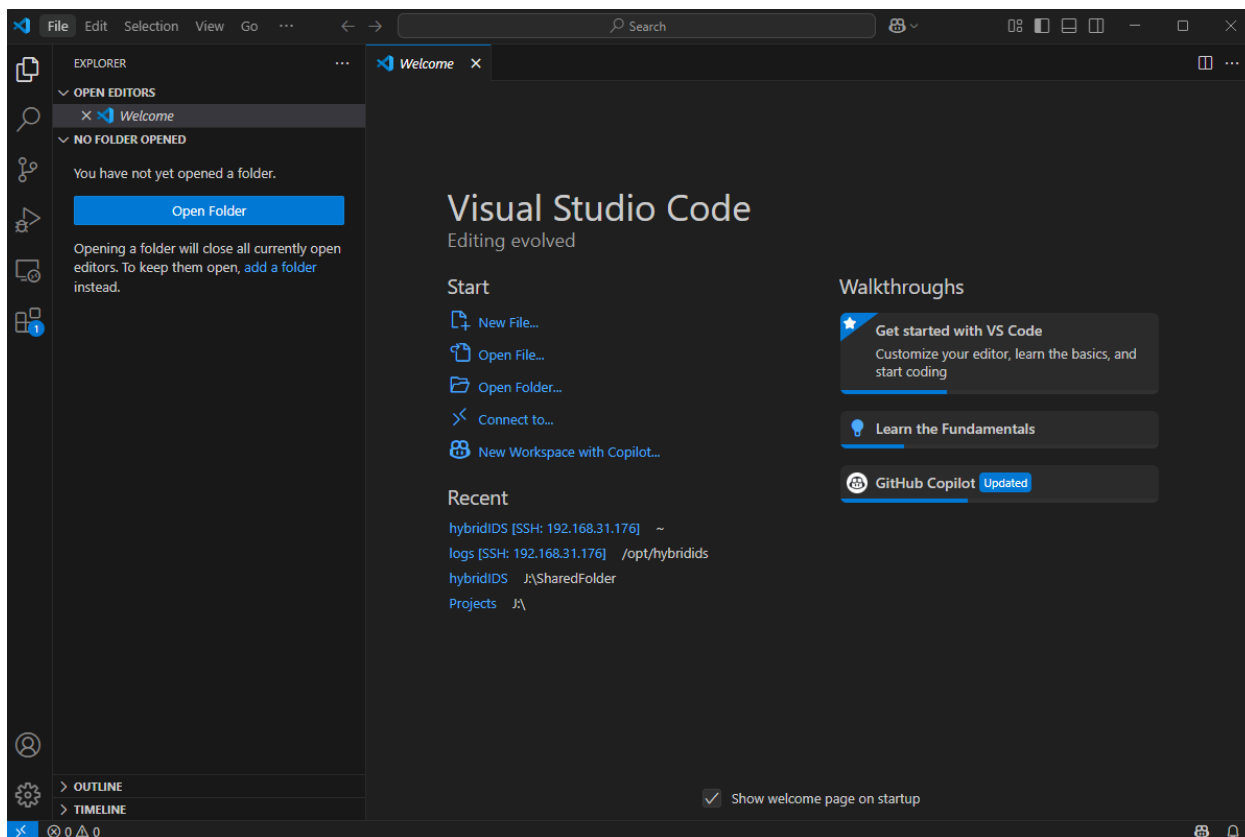


Рисунок 6 – Приветственное окно VSCode

Затем было произведено подключение по SSH к виртуальной машины для удобной работы с кодом. Показано на рисунке 7.

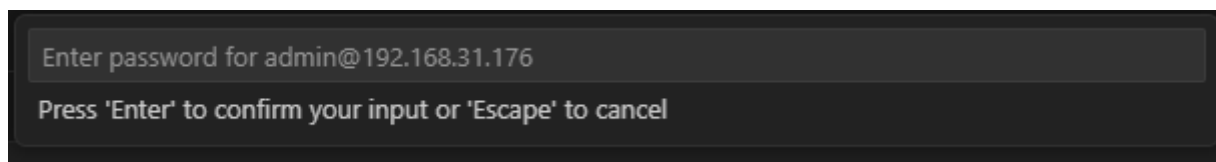


Рисунок 7 – Подключение к серверу

Таким образом получен доступ к файловой системе сервера. Изображено на рисунке 8.

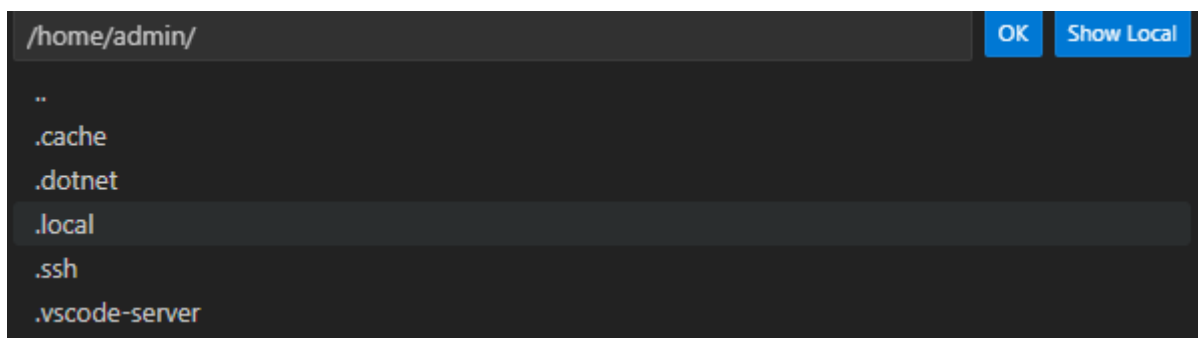


Рисунок 8 – Файловая система VM

3.5 Установка и настройка инструментов в Ubuntu

Для работы с проектом были установлены Docker, conan, cmake, git, curl, nftables, build-essential,

```
# Обновление и установка зависимостей
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential cmake git curl nftables
docker.io
```

```
# Установка Conan
pip3 install conan
```

```
# Настройка nftables (для отладки)
sudo systemctl enable nftables
sudo systemctl start nftables
```

3.6 Работа с удаленным репозиторием

Внутри виртуальной машины и VSCode работа производилась с git через консоль, используя следующие команды:

```
# Добавление и фиксация изменений
git add .
git commit -m "Initial commit"

# Синхронизация с удалённым репозиторием
git remote add origin https://github.com/your-username/hybridIDS.git
git push -u origin main
```

3.7 Вывод

В результате выполнения этапов по выбору, установке и настройке инструментов была подготовлена полнофункциональная среда для разработки, отладки и тестирования проекта hybridIDS. Среда обеспечивает кроссплатформенную разработку, интеграцию с современными DevOps-инструментами и эффективное взаимодействие с системными компонентами Linux.

					ПГУ1.090301.1.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

4 Реализация системы

4.1 Структура проекта

Важная часть реализации – структурированность проекта. Она позволяет упростить модифицирование системы, ее сопровождение и изучение. Так как HybridIDS – это сложный многомодульный проект, то мною была воплощена в жизнь следующая структура, изображенная на рисунке 9.

```
hybridIDS/  
├─ CMakeLists.txt  
├─ Dockerfile  
├─ build.sh  
├─ conanfile.py  
├─ config/  
│   ├─ config.json  
│   ├─ rules.json  
│   └─ whitelist.txt  
├─ include/  
│   └─ *.h  
├─ src/  
│   └─ *.cpp  
├─ tests/  
└─ .vscode/
```

Рисунок 9 – Структура проекта

В корне проекта расположены файлы, отвечающие за сборку, конфигурацию и установку бинарного файла. А именно:

- CMakeLists.txt - конфигурация сборки через CMake;
- Dockerfile - описание контейнера для сборки проекта;
- build.sh - сценарий автоматической сборки и установки;
- conanfile.py – описание зависимостей для Conan.

В директории config расположены все редактируемые файлы для настройки IDS:

- config.json - основные параметры конфигурации;
- rules.json - сигнатурные правила IDS;
- whitelist.txt - белый список разрешённых IP.

В директории include расположены все заголовочные файлы каждого компонента, данный репозиторий нужен для подключения зависимостей в сборке.

В директории src расположены файлы с реализацией классов и методов объявленных в заголовочных файлах.

В директории tests находятся сборочный CMakeLists.txt и файл с реализацией юнит тестов.

В директории .vscode расположена конфигурация среды разработки.

4.2 Основной поток (main)

Основной поток в реализации гибридной системы обнаружения вторжений отвечает за инициализацию всех ключевых компонентов, настройку среды и контроль работы приложения до его завершения. Работа основного потока организована поэтапно и охватывает следующие шаги:

- загрузка конфигурации.

Первым этапом загружается конфигурационный файл:

```
if (!loadConfig(configPath, config))
```

Здесь из config/config.json извлекаются параметры работы системы, такие как интерфейс захвата трафика, уровень логирования и другие настройки. Если конфигурация не загружается, система завершает работу с ошибкой;

- инициализация логгера.

Создается объект Logger, который отвечает за ведение логов всей системы:

```
Logger logger;  
logger.setLogLevel(config.logLevel);  
globalLogger = &logger;
```

Логгер используется как глобальный, чтобы можно было из обработчика сигналов логировать сообщения завершения;

- обработка сигналов завершения.

Устанавливаются обработчики системных сигналов SIGINT и SIGTERM, чтобы обеспечить корректное завершение работы при остановке приложения:

```
std::signal(SIGINT, signalHandler);  
std::signal(SIGTERM, signalHandler);
```

Флаг running используется для контроля цикла жизни программы;

- инициализация и загрузка компонентов.

Создаются и инициализируются основные модули системы: Rule Engine, Heuristic Engine, Nftables Control, Sniffer.

Так же загружаются сигнатурные правила и белый список:

```
ruleEngine.loadRules("config/rules.json");  
ruleEngine.loadWhitelist("config/whitelist.txt");
```

- обработка сетевого трафика.

На компонент Sniffer устанавливается обработчик пакетов:

```
sniffer.setPacketHandler([&](const Packet& pkt, const uint8_t*  
rawData, size_t dataLen)
```

Каждый перехваченный пакет анализируется двумя способами: сигнатурный анализ (ruleEngine.checkPacket) и эвристический анализ (heuristicEngine.analyzePacket).

При обнаружении угрозы вызывается блокировка соответствующего IP-адреса:

```
nftables.blockIP(pkt.srcIP);
```

- запуск и цикл ожидания

После настройки всех компонентов запускается сниффер:

```
sniffer.start();
```

Затем основной поток переходит в цикл ожидания, проверяя флаг running:

```
while (running)  
    std::this_thread::sleep_for(std::chrono::seconds(1));
```

Это позволяет программе работать в фоновом режиме до тех пор, пока не будет получен сигнал завершения;

– корректное завершение работы. После получения сигнала о завершении: останавливается сниффер, логируется завершение работы, освобождаются ресурсы.

```
sniffer.stop();  
logger.log("hybridIDS completed", "main", LogLevel::INFO);
```

Таким образом, основной поток программы организует жизненный цикл системы: от инициализации компонентов до безопасного завершения работы. Он обеспечивает надежное управление обработкой сетевых пакетов, логированием и реагированием на угрозы, делая архитектуру гибридной IDS удобной для масштабирования и сопровождения.

4.3 Модуль работы с конфигурационным файлом (config_utils)

Для обеспечения гибкости и простоты настройки гибридной системы обнаружения вторжений реализован модуль загрузки конфигурации. Он позволяет пользователю задавать ключевые параметры системы через JSON-файл без необходимости пересборки проекта.

Конфигурационные данные описываются в структуре Config, определенной в заголовочном файле config_utils.h:

```
struct Config {  
    std::string interface;  
    LogLevel logLevel = LogLevel::INFO;  
    int alertCooldown = 10;  
    int heuristicPacketThreshold = 100;  
    int heuristicTimeWindowSeconds = 10;  
};
```

– interface – имя сетевого интерфейса, на котором будет вестись захват трафика;

– logLevel – уровень логирования (debug, info, warning, error);

– alertColldown – время в секундах, в течении которого повторное срабатывание предупреждения для одного и того же IP блокируется;

– heuristicPacketThreshold – минимальное количество пакетов за определенное время для срабатывания правила;

– heuristicTimeWindowSeconds – временное окно.

Функция loadConfig реализована в config_utils.cpp и отвечает за считывание и валидацию данных из конфигурационного файла:

```
bool loadConfig(const std::string& fileName, Config& config);
```

Алгоритм работы включает следующие этапы:

- 1) открытие файла;
- 2) ичитывание и разбор JSON с помощью nlohmann/json;
- 3) проверка и загрузка параметров

Каждый параметр проверяется на корректность. Если параметр отсутствует, то используется значение по умолчанию. Например:

```
if (j.contains("log_level") && j["log_level"].is_string())  
    config.logLevel =  
    parseLogLevel(j["log_level"].get<std::string>());
```

Дополнительно реализована функция parseLogLevel, которая преобразует строковое представление уровня логирования в член enum class:

```
static LogLevel parseLogLevel(const std::string& levelStr);
```

Функция учитывает различные варианты записи ("warn", "warning", "debug" и т.д.) и всегда возвращает корректное значение или INFO по умолчанию.

Такой подход позволяет легко изменять настройку системы – например, адаптировать её к различным условиям сети, уровням детализации логов или стратегии реагирования на угрозы.

Модуль загрузки конфигурации делает систему гибкой, удобной и расширяемой. Он отделяет параметры работы от кода и упрощает настройку, тестирование и внедрение гибридной IDS в различных средах.

4.4 Объектно-ориентированное представление сетевого пакета (packet)

Для анализа сетевого трафика и передачи информации между компонентами гибридной IDS разработан компактный и универсальный класс Packet. Он представляет собой абстрактное представление сетевого пакета и содержит поля, необходимые для анализа в сигнатурном движке.

Структура класса Packet

Класс объявлен в заголовочном файле packet.h и реализован в packet.cpp. Он инкапсулирует следующую информацию о каждом перехваченном пакете:

```
class Packet
{
public:
    std::string srcIP;
    std::string dstIP;
    uint16_t srcPort = 0;
    uint16_t dstPort = 0;
    std::string protocol;

    std::string summary() const;
};
```

- srcIP – IP-адрес отправителя;
- dstIP – IP-адрес получателя;
- srcPort – порт источника;
- dstPort – порт назначения;
- protocol – название протокола.

Поля инициализируются по умолчанию, чтобы избежать неопределённого поведения при парсинге пакетов с отсутствующими записями.

Для удобного логирования и отладки реализован метод summary(), возвращающий строку с кратким описанием содержимого пакета:

```
std::string Packet::summary() const
{
    std::ostringstream oss;
    oss << "Protocol: " << protocol
        << ", " << srcIP << ":" << srcPort
```

```

        << " → " << dstIP << ":" << dstPort;
    return oss.str();
}

```

Метод используется для вывода информации о пакете в логах или на экране и оформляет данные в читаемом виде, например:

Protocol: TCP, 192.168.0.5:443 → 192.168.0.12:50500

Назначение и использование:

Экземпляры класса Packet создаются и заполняются на этапе анализа сетевого трафика в модуле Sniffer. Затем они передаются для анализа и могут использоваться для логирования событий.

Класс Packet является ключевым элементом внутреннего представления сетевого трафика в системе.

4.5 Компонент захвата трафика (sniffer)

Одним из ключевых компонентов гибридной системы обнаружения вторжений является модуль захвата сетевого трафика. Для этой цели реализован класс Sniffer, который обеспечивает асинхронный перехват сетевых пакетов с заданного интерфейса, их базовую фильтрацию и парсинг на уровне IP и транспортных протоколов.

Интерфейс класса Sniffer:

```

class Sniffer
{
public:
    using PacketHandler = std::function<void(const Packet&,
    const uint8_t*, size_t)>;

    Sniffer(const std::string& interfaceName, Logger* logger =
    nullptr);
    ~Sniffer();

    void setPacketHandler(PacketHandler handler);
    void start();
    void stop();

private:

```

```

std::string interface;
pcap_t* handle = nullptr;
bool running = false;
PacketHandler packetHandler;
std::thread captureThread;
Logger* logger;

void captureLoop();
};

```

Ключевые элементы:

- interface – имя сетевого интерфейса;
- handle – дескриптор libpcap, используемый для получения пакетов;
- packetHandler – функция-обработчик, вызываемая для каждого перехваченного пакета;
- captureThread – отдельный поток для непрерывного захвата трафика без блокировки основного потока;
- logger – указатель на логгер, используемый для регистрации ошибок и событий.

Захват трафика осуществляется через библиотеку libpcap – де-факто стандарт в UNIX-системах. Захват запускается методом start():

```

void Sniffer::start()
{
    handle = pcap_open_live(interface.c_str(), BUFSIZ, 1, 1000,
errbuf);
    ...
    captureThread = std::thread([this]() { captureLoop(); });
}

```

Захват ведётся в отдельном потоке, что позволяет обрабатывать трафик параллельно с остальными задачами системы.

Метод captureLoop() выполняет основной цикл перехвата и разбора пакетов:

```
int result = pcap_next_ex(handle, &header, &data);
```

Если данные получены, то они анализируются послойно:

- Ethernet: проверяется, что пакет является IPv4;
- IP: извлекаются IP-адреса и тип транспортного протокола;
- TCP/UDP/ICMP: извлекаются и сохраняются в Packet намере портов.

Пример разбора TCP-пакета приведен ниже.

```
const struct tcphdr* tcph = (const struct tcphdr*)(data + ...);  
pkt.srcPort = ntohs(tcph->th_sport);  
pkt.dstPort = ntohs(tcph->th_dport);
```

После успешного разбора вызывается ранее установленный обработчик пакета.

```
if (packetHandler)  
    packetHandler(pkt, data, header->caplen);
```

Это позволяет системе модульно подключать сигнатурный и эвристический анализаторы без жесткой связки с компонентом захвата.

Метод stop() корректно завершает захват, сигнализируя потоку завершить цикл:

```
void Sniffer::stop()  
{  
    running = false;  
    pcap_breakloop(handle);  
    if (captureThread.joinable())  
        captureThread.join();  
}
```

Также производится закрытие pcap-дескриптора в деструкторе, если он был открыт.

При наличии логгера, компонент записывает в журнал ошибки открытия интерфейса, внутренние ошибки pcap_next_ex, и другую отладочную информацию. Это облегчает диагностику во время разработки и эксплуатации.

Класс Sniffer инкапсулирует всю низкоуровневую работу с сетевым трафиком и обеспечивает мощный, расширяемый механизм передачи пакетов на анализ. Его реализация основана на библиотеке libpcap и полностью соответствует принципам многопоточности, модульности и отказоустойчивости.

Этот компонент является критически важным звеном между реальной сетевой активностью и логикой обнаружения угроз в IDS.

4.6 Сигнатурный анализ (rule_engine)

Класс RuleEngine реализует функционал сигнатурного анализа сетевого трафика. Он предназначен для обнаружения подозрительных пакетов на основе заданных правил, автоматической блокировки источников угроз, а также управления белыми списками доверенных IP-адресов.

В системе каждое правило представлено структурой Rule.

```
struct Rule
{
    std::string srcIP;
    std::string dstIP;
    int srcPort = -1;
    int dstPort = -1;
    std::string protocol;

    int icmpType = -1;
    int icmpCode = -1;
};
```

Каждое поле может быть либо конкретизировано, либо оставлено пустым/равным -1 — в этом случае оно игнорируется при проверке.

Интерфейс RuleEngine представлен ниже.

```
class RuleEngine
{
public:
    RuleEngine(const Config& config, Logger* logger = nullptr);
    bool loadRules(const std::string& fileName);
    bool checkPacket(const Packet& packet);
    bool loadWhitelist(const std::string& fileName);

private:
    std::vector<Rule> rules;
    NftablesControl nft;
    std::unordered_set<std::string> blockedIPs;
    std::unordered_set<std::string> whitelist;
    std::unordered_map<std::string, time_t> recentAlerts;
    int alertCooldownSeconds;
    Logger* logger;
};
```

Ключевые элементы:

- rules – список правил фильтрации, загружаемых из JSON-файла;
- nft – интерфейс управления системным брандмауэром nftables;

- blockedIPs – множество уже заблокированных IP-адресов;
- whitelist – множество доверенных IP, исключаемых из фильтрации;
- logger – механизм логирования событий и ошибок;
- recentAlerts и alertCooldownSeconds – механизм предотвращения повторных уведомлений по одному IP с учетом заданного интервала времени;

Загрузка правил:

- метод loadRules отвечает за загрузку и валидацию правил из JSON-файла;
- файл открывается и парсится с помощью библиотеки nlohmann/json;
- ожидается JSON-массив объектов, каждый из которых описывает правило с полями: IP-адреса, порты, протокол, ICMP-тип и код;
- выполняется проверка корректности портов (от -1 до 65535, где -1 — игнорировать порт);
- правила, прошедшие валидацию, сохраняются во внутренний вектор;
- ошибки открытия файла, парсинга или формата логируются с уровнем ERROR.

Проверка пакетов:

- метод checkPacket реализует логику сопоставления пакета с правилами;
- сначала проверяется, не входит ли IP источника в белый список — в таком случае пакет пропускается;
- для каждого правила проверяются условия: совпадение IP-адресов, протокола, портов (для TCP/UDP) или ICMP-типов;
- проверяется временной интервал с момента последнего предупреждения по IP, чтобы не засорять логи;
- если IP не заблокирован, вызывается метод nft.blockIP для добавления правила блокировки на уровне системы;
- IP добавляется в список заблокированных;
- метод возвращает true, если пакет удовлетворил правило и IP был заблокирован, иначе — false;

– метод loadWhitelist загружает IP из текстового файла, где каждый адрес занимает отдельную строку. Это позволяет исключать известные безопасные адреса из анализа.

При наличии объекта Logger, все ключевые действия, ошибки и предупреждения фиксируются в журнал, что облегчает диагностику и сопровождение.

В итоге мы имеем компонент RuleEngine, который реализует гибкий и расширяемый механизм контроля сетевых пакетов, обеспечивая эффективное применение правил фильтрации и интеграцию с системным брандмауэром. Его архитектура основана на производительности, предотвращении повторных срабатываний и удобстве конфигурирования через JSON-файлы. Этот модуль является критическим для обеспечения безопасности сети и быстрого реагирования на потенциальные угрозы.

4.7 Эвристический анализ (heuristic_engine)

Модуль эвристического анализа трафика выполняет функцию оценивания на подозрительность пакетов на основе различных критериев: скорости пакетов, флагов TCP, аномалий портов, размера пакетов и других признаков. Основная задача – выявить потенциально вредоносную активность без жестких сигнатур.

```
class HeuristicEngine
{
public:
    HeuristicEngine(Config& config, Logger* logger = nullptr);

    // Анализирует пакет, возвращает true при обнаружении
    // подозрительной активности
    bool analyzePacket(const Packet& packet, const uint8_t*
        rawData, size_t dataLen);

private:
    Logger* logger;

    struct IpStats
    {
        int packetCount = 0;
        std::chrono::steady_clock::time_point firstPacketTime;
```

```

};

    std::unordered_map<std::string, IpStats> ipStats;           //
    Статистика для TCP и ICMP
    std::unordered_map<std::string, IpStats> ipStatsUDP;       //
    Отдельно для UDP

    const int packetThreshold;                                // Порог количества
    пакетов за окно времени
    const std::chrono::seconds timeWindow; // Размер окна
    времени для подсчёта пакетов

    // Методы проверки различных эвристик
    bool checkPacketRateTCP(const Packet& packet);
    bool checkPacketRateUDP(const Packet& packet);
    bool checkTcpFlags(const Packet& packet, const uint8_t*
rawData, size_t dataLen);
    bool checkPortAnomalies(const Packet& packet);
    bool checkEmptyTcpPacket(const Packet& packet, const
uint8_t* rawData, size_t dataLen);
    bool checkPacketSizeAnomaly(size_t dataLen);
    bool checkEmptyUdpPacket(const Packet& packet, const
uint8_t* rawData, size_t dataLen);
    bool checkUdpPacketSize(size_t dataLen);
    bool checkUdpPortAnomalies(const Packet& packet);
    bool checkPacketRateICMP(const Packet& packet);
    bool checkIcmpAnomalies(const Packet& packet, const uint8_t*
rawData, size_t dataLen);
};

```

Ключевые элементы класса:

- logger — указатель на объект логгера для записи предупреждений и информации о подозрительных событиях;
- ipStats, ipStatsUDP — словари, хранящие статистику о количестве пакетов от каждого IP за период времени (раздельно для TCP/ICMP и UDP);
- packetThreshold — пороговое число пакетов, превышение которого считается подозрительным;
- timeWindow — временной интервал, за который считается количество пакетов;
- несколько приватных методов для проверки различных подозрительных признаков (скорость, TCP-флаги, пустые пакеты, размер, аномальные порты);

– конструктор `HeuristicEngine` инициализируется параметрами конфигурации: порог пакетов и окно времени для анализа;

– главный метод — `analyzePacket()`, вызываемый для каждого пакета определяет протокол (TCP/UDP/ICMP).

Для TCP:

- проверяется скорость отправки пакетов (`checkPacketRateTCP`);
- анализируются флаги TCP (`checkTcpFlags`);
- проверяется наличие пустых TCP-пакетов (`checkEmptyTcpPacket`).

Для UDP:

- анализируется скорость UDP-пакетов (`checkPacketRateUDP`);
- проверяются пустые UDP-пакеты (`checkEmptyUdpPacket`);
- анализируется размер UDP-пакетов (`checkUdpPacketSize`);
- проверяются аномалии портов UDP (`checkUdpPortAnomalies`).

Вне зависимости от протокола проверяются аномалии портов и размер пакета. Для ICMP предусмотрены методы проверки скорости и аномалий. Если одна из проверок возвращает `true` — пакет считается подозрительным.

Детали ключевых методов:

– `checkPacketRateTCP` / `checkPacketRateUDP` / `checkPacketRateICMP`. Отслеживают количество пакетов с одного IP за фиксированное время. Если порог превышен — возвращают `true` и логируют предупреждение;

– `checkTcpFlags`. Анализирует TCP-флаги на наличие несовместимых комбинаций (SYN+FIN, SYN+RST, FIN без ACK и т.п.). Такие аномалии часто указывают на попытки сканирования или атак;

– `checkEmptyTcpPacket` / `checkEmptyUdpPacket`. Проверяют, что полезная нагрузка TCP/UDP не пустая. Пустые пакеты могут быть индикаторами сканирования или DoS-атак;

– checkPortAnomalies / checkUdpPortAnomalies. Проверяют корректность портов (1–65535), а также подозрительный трафик на системные или слишком высокие порты;

– checkPacketSizeAnomaly / checkUdpPacketSize. Проверяют, что размер пакета находится в разумных пределах. Очень маленькие или слишком большие пакеты могут быть подозрительными;

– checkIcmpAnomalies. Анализирует типы ICMP-пакетов (например, пинги или неизвестные типы), что позволяет выявлять подозрительные ICMP-сообщения.

При обнаружении подозрительных признаков все важные события логируются через logger с уровнем предупреждения или информационным уровнем. Это облегчает последующий анализ и отладку работы эвристик.

В итоге получается, что NeuristicEngine – это модуль эвристического анализа сетевого трафика, работающий на уровне отдельных пакетов и базирующийся на множестве проверок, включая анализ скорости трафика, TCP-флагов, портов и размера пакетов. Класс организован с использованием стандартных библиотек C++, поддерживает логирование и легко расширяется.

4.8 Управление сетевыми правилами (nftables_control)

Класс NftablesControl реализует управление сетевыми правилами с использованием библиотеки libmnl и libnftnl для работы с подсистемой Netfilter в Linux через Netlink-сокеты. Основная задача класса – это создание таблиц и цепочек в nftables, а также блокировка IP-адресов.

Интерфейс класса представлен ниже.

```
class NftablesControl
{
public:
    NftablesControl(Logger* logger = nullptr);
    ~NftablesControl();

    bool init();
```

```

    bool blockIP(const std::string& ip);

private:
    struct mnl_socket* nl = nullptr;
    Logger* logger;

    bool ensureTable();
    bool ensureChain();
    bool addRule(const std::string& ip);
};

```

Конструктор и деструктор открывают и закрывают Netlink-сокет. Функция `init()` инициализирует сокет и создает необходимые таблицы и цепочки. Метод `blockIP(ip)` добавляет правило блокировки входящего трафика с указанного IP. Внутренние методы `ensureTable()` и `ensureChain()` создают таблицу и цепочку в `nftables`.

Работа с Netlink-сокетом происходит следующим образом. В конструкторе происходит вызов `init()`, где открывается сокет `mnl_socket_open(NETLINK_NETFILTER)`. Производится привязка сокета к процессу через `mnl_socket_bind`. Создаются таблица и цепочка с помощью внутренних методов. В деструкторе сокет корректно закрывается.

Создание таблицы и цепочки `ensureTable()`:

- создается объект таблицы с помощью `nftnl_table_alloc()`;
- назначается имя таблицы "filter" и семейство протоколов `NFPROTO_INET`;
- формируется Netlink-сообщение с типом `NFT_MSG_NEWTABLE`;
- отправляется команда ядру через сокет;
- проверяется успешность операции.

`ensureChain()`:

- создается объект цепочки `nftnl_chain_alloc()`;
- связывается с таблицей "filter", цепочкой "input" и семейством `NFPROTO_INET`;

– устанавливается `hook` — точка применения правил (`NF_INET_LOCAL_IN`), приоритет, тип цепочки и политика (разрешить трафик по умолчанию);

- формируется Netlink-сообщение с типом `NFT_MSG_NEWCHAIN`;
- отправляется ядру и проверяется успешность;
- добавление правила блокировки IP;
- создается правило с помощью `nftnl_rule_alloc()`;
- привязывается к таблице "filter" и цепочке "input";
- устанавливается семейство протоколов `NFPROTO_INET`;
- добавляются выражения: `payload` – извлечение 4 байт IP-адреса из сетевого заголовка (`offset 12`, длина 4), `cmp` – сравнение извлеченного IP с указанным в параметрах, `immediate` – действие "drop" (блокировка);

– формируется и отправляется Netlink-сообщение с правилом (`NFT_MSG_NEWRULE`);

– возвращается результат операции.

```
Logger logger;  
NftablesControl nft(&logger);  
nft.blockIP("192.168.1.100");
```

Реализованный класс `NftablesControl` обеспечивает низкоуровневое управление правилами `nftables` через Netlink, позволяя динамически создавать таблицы, цепочки и блокировать IP-адреса. Использование библиотеки `libmnl` и `libnftnl` позволяет взаимодействовать с ядром Linux на уровне протокола Netlink, что делает решение гибким и эффективным для задач сетевой фильтрации.

4.9 Журналирование (logger)

Класс `Logger` реализует функционал логирования сообщений с различными уровнями важности в файл и консоль. Он обеспечивает потокобезопасность записи логов, а также поддержку уровней логирования для фильтрации сообщений.

Интерфейс класса приводится ниже.

```
#pragma once

#include <fstream>
#include <mutex>
#include <string>
#include <chrono>
#include <unordered_map>

enum class LogLevel
{
    DEBUG = 0,
    INFO,
    WARNING,
    ERROR
};

class Logger
{
public:
    Logger(std::chrono::seconds floodInterval =
std::chrono::seconds(5), const std::string& filename =
"log/hybridIDS.log");
    ~Logger();

    virtual void log(const std::string& message, const
std::string& module = "general", LogLevel level =
LogLevel::INFO);
    void setLogLevel(LogLevel level);
    static std::string levelToString(LogLevel level);

private:
    std::ofstream logFile;
    std::mutex logMutex;
    LogLevel currentLevel = LogLevel::INFO;

    std::string lastMessage;
    std::chrono::steady_clock::time_point lastLogTime;
    std::chrono::seconds floodInterval;
    std::unordered_map<std::string,
std::chrono::steady_clock::time_point> floodMap;

    std::string getTimestamp() const;
};
```

Ключевые моменты:

- конструктор: открывает файл лога для дозаписи;
- деструктор: корректно закрывает файл лога;

- log(): Основной метод записи сообщений с указанием модуля и уровня важности;
- setLogLevel(): устанавливает минимальный уровень логирования;
- levelToString(): преобразует уровень логирования в строку;
- внутренний метод getTimestamp() формирует временную метку для каждой записи;
- Производится защита от спама в лог-журнале.

Уровни логирования:

```
enum class LogLevel
{
    DEBUG = 0,
    INFO,
    WARNING,
    ERROR
};
```

Они определяют четыре уровня от детального DEBUG до критического ERROR. Таким образом реализована фильтрация сообщений, где выводятся только те, которые соответствуют определенному уровню, заданному в конфигурации.

При создании объекта Logger открывается файл в режиме записи (std::ios::app). Если файл не открылся, выводится сообщение об ошибке в консоль. При уничтожении объекта происходит закрытие файла.

Метод log() проверяет, соответствует ли уровень сообщения текущему уровню логирования. Он использует std::lock_guard<std::mutex> для защиты записи от одновременного доступа из разных потоков. В итоге он формирует строку сообщения в формате:

```
[YYYY-MM-DD HH:MM:SS] [LEVEL] [module] message
```

Далее записывает сообщение в файл и выводит его на стандартный вывод.

Метод getTimestamp() получает текущее системное время. Затем преобразует время в строку формата ГГГГ-ММ-ДД ЧЧ:ММ:СС с помощью strftime. Используется для маркировки каждой записи точным временем.

Метод `setLogLevel()` позволяет изменить текущий минимальный уровень логов. Сообщения с уровнем ниже текущего игнорируются.

Метод `levelToString()` реализует преобразование уровня в удобочитаемую строку, например, `DEBUG`, `INFO`.

Пример использования:

```
Logger logger("logs/hybridIDS.log");
logger.setLogLevel(LogLevel::DEBUG);

logger.log("Система запущена", "system", LogLevel::INFO);
logger.log("Получено подозрительное событие", "IDS",
LogLevel::WARNING);
logger.log("Ошибка записи в базу данных", "database",
LogLevel::ERROR);
```

Реализованный класс `Logger` предоставляет удобный и потокобезопасный механизм для ведения логов в проекте. Поддержка уровней логирования и форматированных временных меток, обеспечивает эффективный мониторинг и отладку работы системы.

4.10 Автоматизированная сборка (build.sh)

В рамках реализации проекта был разработан автоматизированный скрипт `build.sh`, обеспечивающий последовательный процесс сборки, тестирования, установки и настройки системы `HybridIDS`.

Скрипт написан на языке `Bash` и использует контейнеризацию `Docker` для изоляции среды сборки. Это позволяет гарантировать воспроизводимость и минимизировать влияние окружения хоста на результат компиляции[8]. В основе лежит `Docker`-образ `hybridids-build`, который содержит все необходимые зависимости и инструменты.

```
IMAGE_NAME="hybridids-build"
docker build -t $IMAGE_NAME .
```

Скрипт сначала собирает `Docker`-образ на основе локального `Dockerfile`. Далее внутри контейнера происходит очистка старой сборки, установка зависимостей через пакетный менеджер `Conan`, конфигурация проекта с помощью

CMake и непосредственно компиляция исходного кода. После сборки автоматически запускаются юнит-тесты с подробным выводом, что обеспечивает проверку корректности и стабильности кода.

```
docker run --rm -it \
-v "$PWD":/app \
-w /app \
$IMAGE_NAME \
bash -c "
    rm -rf build
    conan install . --output-folder=build --build=missing
    cmake -S . -B build \
    -
    DCMAKE_TOOLCHAIN_FILE=build/build/Release/generators/conan_toolc
hain.cmake \
    -DCMAKE_BUILD_TYPE=Release \
    -DENABLE_TESTS=ON
    cmake --build build
    cd build
    ctest -V --output-on-failure
"
```

По завершении сборки скрипт копирует скомпилированный бинарный файл и конфигурационные файлы в целевую директорию /opt/hybridids. Создаются необходимые каталоги для бинарников, конфигураций и логов.

```
sudo mkdir -p /opt/hybridids/bin
sudo mkdir -p /opt/hybridids/config
sudo mkdir -p /opt/hybridids/logs

sudo cp "$PROJECT_DIR/bin/hybridIDS" "/opt/hybridids/bin/"
sudo cp -r "$PROJECT_DIR/config/"* "/opt/hybridids/config/"
```

Автоматически формируется unit-файл для systemd, описывающий сервис HybridIDS с настройками автозапуска, перезапуска при сбоях и рабочей директорией.

```
sudo bash -c "cat > /etc/systemd/system/hybridids.service <<EOF
[Unit]
Description=HybridIDS Intrusion Detection Service
After=network.target

[Service]
Type=simple
User=root
Group=root
WorkingDirectory=/opt/hybridids
ExecStart=/opt/hybridids/bin/hybridIDS"
```

```
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
"
```

Выполняется перезагрузка конфигурации systemd и включение сервиса в автозагрузку. Скрипт выводит пользователю подробные сообщения о текущем статусе и завершающих шагах, включая инструкции по запуску сервиса и просмотру логов. По окончании предлагается очистить временные файлы сборки для поддержания порядка в проекте, что упрощает дальнейшую работу с исходниками.

```
read -p "[build.sh]: Clear temporary build files (build/, bin/ and
etc.)? [y/N] " confirm

if [[ "$confirm" =~ ^[Yy]$ ]]; then
    sudo rm -rf build/ bin/ CMakeUserPresets.json .Dockerfile.swo
.Dockerfile.swp
fi
```

Разработанный скрипт обеспечивает полный цикл сборки и развертывания системы HybridIDS в стандартизированной и автоматизированной форме. Такой подход значительно повышает удобство эксплуатации и снижает вероятность ошибок, связанных с ручным выполнением команд. Использование Docker и systemd гарантирует стабильность и управляемость сервиса в реальных условиях эксплуатации.

5 Тестирование

5.1 Unit-тесты

Тестирование программного обеспечения является неотъемлемой частью процесса разработки, обеспечивающей корректность и стабильность работы системы[7]. В данной главе будут рассмотрены подходы к тестированию компонентов сетевого приложения с использованием библиотеки Google Test и Google Mock. Я проведу анализ различных типов юнит-тестов, которые обеспечивают проверку функциональности отдельных компонентов системы.

Тесты охватывают ключевые модули, включая загрузку конфигурации, логирование, эвристический анализ пакетов, правила фильтрации, взаимодействие с nftables, а также сетевой сниффер.

Каждый модуль тестировался независимо. Для журналирования использовался мокированный логгер, чтобы перехватывать сообщения без вывода в консоль или файл.

5.1.1 Тестирование модуля конфигурации

Модуль конфигурации отвечает за загрузку параметров работы системы из JSON-файла. Были реализованы следующие тесты:

- LoadValidConfig – проверка успешной загрузки конфигурационного файла;
- LoadInvalidConfig – проверка поведения при попытке загрузить файл с некорректной структурой;
- LoadMissingFile – проверка поведения при отсутствии конфигурационного файла.

Результаты подтвердили устойчивость модуля к ошибочным данным и корректную инициализацию параметров системы.

Пример тестов:

```
TEST_F(ConfigUtilsTest, LoadValidConfig) {  
    Config config;
```

```

    ASSERT_TRUE(loadConfig(validFile, config));
    EXPECT_EQ(config.interface, "eth0");
    EXPECT_EQ(config.logLevel, LogLevel::DEBUG);
    EXPECT_EQ(config.alertCooldown, 5);
    EXPECT_EQ(config.heuristicPacketThreshold, 50);
    EXPECT_EQ(config.heuristicTimeWindowSeconds, 5);
}

TEST_F(ConfigUtilsTest, LoadInvalidConfig) {
    Config config;
    ASSERT_FALSE(loadConfig(invalidFile, config));
}

TEST_F(ConfigUtilsTest, LoadMissingFile) {
    Config config;
    ASSERT_FALSE(loadConfig("nonexistent.json", config));
}

```

5.1.2 Тестирование эвристического анализа

Эвристический движок анализирует трафик и выявляет потенциально вредоносные паттерны на основе содержимого и частоты пакетов. Были проведены следующие проверки:

- PacketRateThresholdTCP – проверяет, что при превышении пороговой частоты TCP-пакетов от одного источника срабатывает тревога;
 - EmptyUdpPacketTriggersAlert – проверяет реакцию на UDP-пакеты без полезной нагрузки;
 - SuspiciousTcpFlagsDetected – проверяет обнаружение TCP-пакетов с подозрительными флагами (например, одновременно SYN и FIN);
 - EmptyTcpPayloadDetected – тревога при пустой TCP;
 - IcmpPacketTriggersAlertOnEmptyPayload
- и IcmpPacketWithSuspiciousTypeTriggersAlert – тесты на ICMP-пакеты с пустыми телами и подозрительными типами.

Эти тесты подтверждают способность эвристик выявлять как признаки сканирования, так и попытки обфускации трафика.

5.1.3 Тестирование сигнатурного анализа

Данный компонент анализирует пакеты на соответствие заранее заданным правилам или белому списку. Проверки включали:

- LoadValidRules / LoadInvalidRulesFile — загрузка и обработка файлов с правилами;
- LoadWhitelist – загрузка белого списка IP-адресов.
- CheckPacketMatchesRule / DoesNotMatchRule – проверка совпадения и несовпадения с заданными правилами;
- WhitelistSkipsDetection – проверка приоритета белого списка над правилами;
- CheckIcmpPacketMatchesRule / DoesNotMatchRule / Whitelisted – то же самое, но для ICMP-трафика;

Модуль демонстрирует корректную фильтрацию и правильную обработку исключений.

5.1.4 Тестирование сниффера

Сниффер отвечает за захват сетевых пакетов и передачу их на анализ. Были реализованы следующие тесты:

- ConstructAndSetHandler – проверка установки обработчика пакетов;
- StartStopSnifferNoCrash – подтверждение корректного запуска и остановки потока захвата;
- InvalidInterfaceLogsError – проверка реакции на несуществующий интерфейс;
- ParseIcmpPacket – проверка корректности разбора простого ICMP-пакета.

Тестирование подтверждает базовую работоспособность и устойчивость модуля к ошибкам интерфейса.

5.1.5 Тестирование nftables-модуля

Этот модуль управляет правилами блокировки IP-адресов. Были проведены тесты:

- Initialization – проверка успешной инициализации взаимодействия с nft;
- BlockValidIP – проверка добавления валидного IP-адреса в черный список;
- BlockInvalidIP – проверка обработки некорректных IP-адресов (например, 256.256.256.256).

Тесты подтверждают устойчивость модуля к ошибочным данным.

5.1.6 Тестирование модуля журналирования

Модуль логирования записывает сообщения в файл или консоль. Были протестированы следующие аспекты:

- LogWritesToFileAndConsole – проверка записи сообщений с указанием уровня и модуля;
- LogLevelFiltering – проверка фильтрации сообщений по уровню логирования;
- TimestampFormat – подтверждение корректного формата временной метки в сообщении;
- LevelToStringReturnsCorrectString – проверка сопоставления уровней логов строковым представлениям.

Результаты показывают корректную фильтрацию и форматирование логов.

5.1.7 Общие выводы и результаты

Проведённое модульное тестирование охватило ключевые компоненты системы. Все тесты выполнялись успешно, ошибки или сбои не были зафиксированы. Отдельное внимание уделялось:

- устойчивости к неверным данным;
- корректной обработке сетевых протоколов;
- адекватной работе логики правил и эвристик;
- взаимодействию с системными утилитами (nftables).

Таким образом, система прошла базовую проверку надежности, функциональности и безопасности. Тесты обеспечивают высокую уверенность в корректности работы как отдельных модулей, так и всей системы в целом. Даже при модификации или доработки системы, разработчик не сможет собрать программу, так как юнит тесты проверяют каждый компонент во время сборки.

Юнит тестирование интегрируется в общую кодовую базу с помощью своего отдельного CMakeLists.txt:

```
find_package(GTest REQUIRED)
find_package(nlohmann_json REQUIRED)
find_package(PkgConfig REQUIRED)

pkg_check_modules(PCAP REQUIRED libpcap)
pkg_check_modules(LIBMNL REQUIRED libmnl)
pkg_check_modules(LIBNFTNL REQUIRED libnftnl)

add_executable(unit_tests
    ../src/logger.cpp
    ../src/sniffer.cpp
    ../src/nftables_control.cpp
    ../src/rule_engine.cpp
    ../src/heuristic_engine.cpp
    ../src/config_utils.cpp
    ../src/packet.cpp
    unit_tests.cpp
)

target_compile_definitions(unit_tests PRIVATE TESTING)

target_include_directories(unit_tests PRIVATE
    ${PROJECT_SOURCE_DIR}/include
    ${PCAP_INCLUDE_DIRS}
    ${LIBMNL_INCLUDE_DIRS}
    ${LIBNFTNL_INCLUDE_DIRS}
)

target_link_libraries(unit_tests
    GTest::gtest
    GTest::gtest_main
    nlohmann_json::nlohmann_json
    pthread

    ${PCAP_LIBRARIES}
    ${LIBMNL_LIBRARIES}
    ${LIBNFTNL_LIBRARIES}
)
```

```
add_test(NAME AllUnitTests COMMAND unit_tests --gtest_color=yes -  
-gtest_print_time=1)
```

Здесь подключаются нужные исходники, библиотеки и зависимости.

5.2 Практическое тестирование

После реализации механизма фильтрации трафика, необходимо убедиться в его корректной работе. Для этого проводится практическое тестирование системы с использованием контролируемых атак, имитирующих угрозы. Цель данного этапа – проверить, насколько эффективно программа обнаруживает и блокирует подозрительные пакеты, соответствующие правилам, заданным в конфигурационном файле.

В ходе экспериментов применяются утилиты низкоуровневой генерации пакетов, такие как hping3, nmap, ping, позволяющие вручную задать параметры IP- и TCP/UDP-пакетов. Особое внимание уделяется проверке обработки таких параметров, как IP-адрес источника, порты, тип протокола и специфические значения ICMP, флаги TCP.

Результаты тестирования позволят выявить возможные уязвимости в логике обработки трафика, а также оценить устойчивость системы к внешним воздействиям и способность своевременно реагировать на подозрительные действия.

5.2.1 Тестирование эвристики

Эмуляция запроса на DNS-сервер без полезной нагрузки. Тестирует реакцию системы на пустой UDP-запрос к стандартному порту 53, который часто используется в DDoS-атаках методом amplification. Визуализация представлена на рисунках 10 и 11.

```

admin@Attacker:~$ sudo hping3 --udp -p 53 --data 0 192.168.31.176
HPING 192.168.31.176 (enp0s3 192.168.31.176): udp mode set, 28 headers + 0 data
bytes
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1547 seq=0
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1548 seq=1
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1549 seq=2
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1550 seq=3
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1551 seq=4
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1552 seq=5

```

Рисунок 10 – Запрос без полезной нагрузки

```

2025-06-06 21:22:24][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
2025-06-06 21:22:24][INFO][nftables] IP blocking via nftables: 192.168.31.249
2025-06-06 21:22:29][WARNING][heuristic_engine] Heuristic alert: Traffic to high port 62757 from IP 192.168.31.176
2025-06-06 21:22:33][INFO][nftables] IP blocking via nftables: 192.168.31.176
2025-06-06 21:22:34][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
2025-06-06 21:22:34][INFO][nftables] IP blocking via nftables: 192.168.31.249
2025-06-06 21:22:39][WARNING][heuristic_engine] Heuristic alert: Traffic to high port 62757 from IP 192.168.31.176
2025-06-06 21:22:42][WARNING][heuristic_engine] Heuristic alert: Empty UDP packet from IP 192.168.31.192

```

Рисунок 11 – Реакция на запрос без полезной нагрузки

Имитация SYN-флуда – распространённой формы DDoS-атаки, перегружающей сервер множеством незавершённых TCP-соединений. Система должна обнаружить частоту отправки пакетов и заблокировать источник. Визуализация представлена на рисунках 12 и 13.

```

admin@Attacker:~$ sudo hping3 -S -p 80 --flood 192.168.31.176
HPING 192.168.31.176 (enp0s3 192.168.31.176): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.31.176 hping statistic ---
2002255 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```

Рисунок 12 – SYN-Flood

```

[2025-06-06 21:21:17][INFO][rule_engine] Loaded whitelist from file: config/whitelist.txt
[2025-06-06 21:21:18][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.176
[2025-06-06 21:21:18][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:21:18][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
[2025-06-06 21:21:18][INFO][nftables] IP blocking via nftables: 192.168.31.249
[2025-06-06 21:21:28][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:21:28][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:21:28][WARNING][rule_engine] IP already blocked: 192.168.31.176
[2025-06-06 21:21:28][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:21:33][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
[2025-06-06 21:21:33][INFO][nftables] IP blocking via nftables: 192.168.31.249
[2025-06-06 21:21:36][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 140.82.114.22
[2025-06-06 21:21:36][INFO][nftables] IP blocking via nftables: 140.82.114.22
[2025-06-06 21:21:38][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:21:38][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:21:38][WARNING][rule_engine] IP already blocked: 192.168.31.176
[2025-06-06 21:21:38][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:21:44][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
[2025-06-06 21:21:44][INFO][nftables] IP blocking via nftables: 192.168.31.249
[2025-06-06 21:21:48][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:21:48][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:21:48][WARNING][heuristic_engine] Heuristic alert: Traffic to high port 63679 from IP 192.168.31.176
[2025-06-06 21:21:48][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:21:58][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:21:58][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:21:58][WARNING][rule_engine] IP already blocked: 192.168.31.176
[2025-06-06 21:21:58][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:22:03][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.249
[2025-06-06 21:22:03][INFO][nftables] IP blocking via nftables: 192.168.31.249

```

Рисунок 13 – Реакция сервера на SYN-Flood

Тестирование реакции на аномальные комбинации TCP-флагов. Пакеты с флагами SYN и FIN одновременно не имеют смысла и часто используются для обхода фильтров. Визуализация представлена на рисунках 14 и 15.

```

admin@Attacker:~$ sudo hping3 192.168.31.176 -p 80 -SF -c 1
HPING 192.168.31.176 (enp0s3 192.168.31.176): SF set, 40 headers + 0 data bytes
len=46 ip=192.168.31.176 ttl=64 DF id=0 sport=80 flags=RA seq=0 win=0 rtt=0.8 ms

--- 192.168.31.176 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.8/0.8/0.8 ms
admin@Attacker:~$

```

Рисунок 14 – Аномальные комбинации флагов

```

[2025-06-06 21:32:50][WARNING][heuristic_engine] Heuristic alert: Traffic to high port 62737 from IP 192.168.31.176
[2025-06-06 21:33:00][WARNING][heuristic_engine] Heuristic alert: Suspicious TCP flags SYN+FIN or SYN+RST from IP 192.168.31.192
[2025-06-06 21:33:00][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:33:00][INFO][nftables] IP blocking via nftables: 192.168.31.176

```

Рисунок 15 – Блокировка подозрительных флагов

Проверка реакции на пакеты АСК без предварительно установленного соединения. Может использоваться для разведки или сканирования портов. Изображено на рисунках 16 и 17.

```
admin@Attacker:~$ sudo hping3 192.168.31.176 -p 80 -A -c 1
HPING 192.168.31.176 (enp0s3 192.168.31.176): A set, 40 headers + 0 data bytes
len=46 ip=192.168.31.176 ttl=64 DF id=0 sport=80 flags=R seq=0 win=0 rtt=1.8 ms

--- 192.168.31.176 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.8/1.8/1.8 ms
admin@Attacker:~$
```

Рисунок 16 – Отправка АСК-пакетов

```
[2025-06-06 21:34:23][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:34:23][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:34:23][WARNING][rule_engine] Suspicious package detected: Protocol: TCP, 192.168.31.176:80 → 192.168.31.192:2433
[2025-06-06 21:34:23][INFO][nftables] IP blocking via nftables: 192.168.31.176
"/opt/hybridids/logs/hybridIDS.log" [readonly] 236L, 22287B
```

Рисунок 17 – Реакция на АСК-пакеты

Генерация потока UDP-пакетов с высокой частотой 500 пакетов в секунду. Проверяет способность системы обнаруживать и блокировать флуд-атаки по UDP. Показано на рисунках 18 и 19.

```
admin@Attacker: ~
SENT (1.2102s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2112s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2122s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2132s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2142s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2152s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2162s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2173s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2183s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2193s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2203s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2213s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2224s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2234s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2244s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2254s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2264s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2274s) UDP packet with 5 bytes to 192.168.31.176:40125
SENT (1.2284s) UDP packet with 5 bytes to 192.168.31.176:40125

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
UDP packets sent: 1000 | Rcvd: 0 | Lost: 1000 (100.00%)
Nping done: 1 IP address pinged in 2.23 seconds
admin@Attacker:~$
```

Рисунок 18 – Генерация потока UDP-пакетов

```

2025-06-06 21:34:23[INFO][nftables] IP blocking via nftables: 192.168.31.176
2025-06-06 21:34:53[WARNING][heuristic_engine] Heuristic alert: Traffic to high port 62757 from IP 192.168.31.176
2025-06-06 21:35:05[WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 192.168.31.176
2025-06-06 21:35:05[WARNING][heuristic_engine] Heuristic alert: Traffic to high port 59672 from IP 92.63.177.253
2025-06-06 21:36:41[WARNING][heuristic_engine] Heuristic alert: High UDP packet rate from IP 192.168.31.192
2025-06-06 21:36:41[INFO][nftables] IP blocking via nftables: 192.168.31.192
/opt/hybridids/logs/hybridIDS.log" [readonly] 241L, 22823B

```

Рисунок 19 – Реакция на поток UDP-пакетов

Повторная отправка небольших UDP-запросов на порт 53 – распространённая техника DDoS. Проверяется фильтрация по порту и объёму данных. Визуализация представлена на рисунках 20 и 21.

```

admin@Attacker:~$ sudo hping3 --udp -p 53 192.168.31.176 -c 10 -d 10
HPING 192.168.31.176 (enp0s3 192.168.31.176): udp mode set, 28 headers + 10 data
bytes
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1468 seq=0
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1469 seq=1
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1470 seq=2
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1471 seq=3
ICMP Port Unreachable from ip=192.168.31.176 name=HybridIDS
status=0 port=1472 seq=4
^C
--- 192.168.31.176 hping statistic ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.5/2.0/4.5 ms
admin@Attacker:~$

```

Рисунок 20 – Отправка запросов

```

2025-06-06 21:36:41[WARNING][heuristic_engine] Heuristic alert: High UDP packet rate from IP 192.168.31.192
2025-06-06 21:36:41[INFO][nftables] IP blocking via nftables: 192.168.31.192
2025-06-06 21:36:56[WARNING][heuristic_engine] Heuristic alert: Traffic to high port 62757 from IP 192.168.31.176
2025-06-06 21:38:07[WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 192.168.31.192
/opt/hybridids/logs/hybridIDS.log" [readonly] 243L, 23055B

```

Рисунок 21 – Реакция на запросы

Имитация ICMP-флуда. Тест направлен на проверку обнаружения чрезмерной активности с использованием Echo-запросов. Представлено на рисунках 22 и 23.

```

admin@Attacker: ~
) id=35655 seq=997] IP [ttl=64 id=44203 iplen=28 ]
RCVD (2.0172s) ICMP [192.168.31.176 > 192.168.31.192 Echo reply (type=0/code=0)
id=35655 seq=997] IP [ttl=64 id=48614 iplen=28 ]
SENT (2.0191s) ICMP [192.168.31.192 > 192.168.31.176 Echo request (type=8/code=0)
) id=35655 seq=998] IP [ttl=64 id=44203 iplen=28 ]
RCVD (2.0193s) ICMP [192.168.31.176 > 192.168.31.192 Echo reply (type=0/code=0)
id=35655 seq=998] IP [ttl=64 id=48617 iplen=28 ]
SENT (2.0211s) ICMP [192.168.31.192 > 192.168.31.176 Echo request (type=8/code=0)
) id=35655 seq=999] IP [ttl=64 id=44203 iplen=28 ]
RCVD (2.0213s) ICMP [192.168.31.176 > 192.168.31.192 Echo reply (type=0/code=0)
id=35655 seq=999] IP [ttl=64 id=48618 iplen=28 ]
SENT (2.0231s) ICMP [192.168.31.192 > 192.168.31.176 Echo request (type=8/code=0)
) id=35655 seq=999] IP [ttl=64 id=44203 iplen=28 ]
RCVD (2.0234s) ICMP [192.168.31.176 > 192.168.31.192 Echo reply (type=0/code=0)
id=35655 seq=999] IP [ttl=64 id=48620 iplen=28 ]
SENT (2.0252s) ICMP [192.168.31.192 > 192.168.31.176 Echo request (type=8/code=0)
) id=35655 seq=1000] IP [ttl=64 id=44203 iplen=28 ]
RCVD (2.0253s) ICMP [192.168.31.176 > 192.168.31.192 Echo reply (type=0/code=0)
id=35655 seq=1000] IP [ttl=64 id=48621 iplen=28 ]

Max rtt: 0.885ms | Min rtt: 0.018ms | Avg rtt: 0.096ms
Raw packets sent: 1000 (28.000KB) | Rcvd: 1000 (46.000KB) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 2.06 seconds
admin@Attacker:~$

```

Рисунок 22 – ICMP-Flood

```

[2025-06-06 21:38:58][WARNING]heuristic_engine1 Heuristic alert: Traffic to high port 62757 from IP 192.168.31.192
[2025-06-06 21:38:59][WARNING]heuristic_engine1 Heuristic alert: UDP traffic on system port from IP 192.168.31.192
[2025-06-06 21:39:26][INFO]heuristic_engine1 Heuristic alert: Traffic to high port 62757 from IP 192.168.31.176
[2025-06-06 21:39:26][INFO]heuristic_engine1 Heuristic alert: ICMP Echo Request from IP 192.168.31.192
[2025-06-06 21:39:26][WARNING]heuristic_engine1 Heuristic alert: High ICMP packet rate from IP 192.168.31.192
[2025-06-06 21:39:26][WARNING]heuristic_engine1 Heuristic alert: High ICMP packet rate from IP 192.168.31.176
[2025-06-06 21:39:26][INFO]nftables1 IP blocking via nftables: 192.168.31.176

```

Рисунок 23 – Реакция на ICMP-Flood

Отправка ICMP-пакета с минимальной полезной нагрузкой. Тест проверяет реакцию системы на нестандартно малые запросы. Показано на рисунках 24 и 25.

```

admin@Attacker:~$ sudo hping3 192.168.31.176 -c 1 -d 1
HPING 192.168.31.176 (enp0s3 192.168.31.176): NO FLAGS are set, 40 headers + 1 data bytes
len=46 ip=192.168.31.176 ttl=64 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=2.4 ms

--- 192.168.31.176 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.4/2.4/2.4 ms
admin@Attacker:~$

```

Рисунок 24 – Минимальная полезная нагрузка пакета

```
[2025-06-06 21:39:26][WARNING][heuristic_engine] Heuristic alert: High ICMP packet rate from IP 192.168.31.192
[2025-06-06 21:39:26][WARNING][heuristic_engine] Heuristic alert: High ICMP packet rate from IP 192.168.31.176
[2025-06-06 21:39:26][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:41:23][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.176
[2025-06-06 21:41:23][INFO][nftables] IP blocking via nftables: 192.168.31.176
"/opt/hybridids/logs/hybridIDS.log" [readonly] 250L, 23761B
```

Рисунок 25 – Реакция на пакеты с минимальной нагрузкой

Имитация oversized ICMP-пакета. Данный тест проверяет обработку пакетов, которые могут привести к фрагментации или переполнению буфера. Изображено на рисунках 26 и 27.

```
admin@Attacker:~$ sudo hping3 192.168.31.176 -c 1 -d 20000
HPING 192.168.31.176 (enp0s3 192.168.31.176): NO FLAGS are set, 40 headers + 200
00 data bytes
len=46 ip=192.168.31.176 ttl=64 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=2.8 ms

--- 192.168.31.176 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.8/2.8/2.8 ms
admin@Attacker:~$
```

Рисунок 26 – Имитация большого ICMP-пакета

```
06-06 21:42:32][INFO][nftables] IP blocking via nftables: 192.168.31.192
06-06 21:42:32][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 1514
06-06 21:42:32][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 1514
06-06 21:42:32][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 1514
06-06 21:42:32][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 1514
06-06 21:42:32][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 1514
hybridids/logs/hybridIDS.log" [readonly] 272L, 25773B
```

Рисунок 26 – Ответ на большой ICMP-пакет

5.2.2 Тестирование сигнатурного анализа

Классический stealth-скан, выполняемый с отправкой TCP SYN-пакетов на стандартные порты. При этом не устанавливается полноценное соединение, что делает сканирование менее заметным. Здесь производится проверка способности системы отследить попытку скрытого TCP-сканирования и зафиксировать подозрительную активность. Представлено на рисунках 27 и 28.

```

admin@Attacker:~$ sudo nmap -sS -p 22,80,443 192.168.31.176
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 21:53 UTC
Nmap scan report for HybridIDS (192.168.31.176)
Host is up (0.00022s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    closed http
443/tcp    closed https
MAC Address: 08:00:27:5F:A0:45 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
admin@Attacker:~$ nmap -sU -p 22,80,443 192.168.31.176
You requested a scan type which requires root privileges.
QUITTING!
admin@Attacker:~$ sudo nmap -sU -p 22,80,443 192.168.31.176
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 21:54 UTC
Nmap scan report for HybridIDS (192.168.31.176)
Host is up (0.00031s latency).

PORT      STATE SERVICE
22/udp    closed ssh
80/udp    closed http
443/udp    closed https
MAC Address: 08:00:27:5F:A0:45 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
admin@Attacker:~$

```

Рисунок 27 – TCP/UDP сканирование

```

[2025-06-06 21:47:12][WARNING][heuristic_engine] Heuristic alert: Very large packet size: 4473
[2025-06-06 21:53:37][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.192
[2025-06-06 21:53:37][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:53:37][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.176
[2025-06-06 21:53:37][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:54:23][WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 192.168.31.192
[2025-06-06 21:54:23][INFO][nftables] IP blocking via nftables: 192.168.31.192

```

Рисунок 28 – Реакция на TCP/UDP сканирование

Проверка, правильно ли система обрабатывает легитимные, одиночные ICMP-запросы без блокировки или ложных срабатываний. Блокировки не происходит, лишь предупреждение. Представлено на рисунках 29 и 30.

```
admin@Attacker:~$ ping -c 3 192.168.31.176
PING 192.168.31.176 (192.168.31.176) 56(84) bytes of data.
64 bytes from 192.168.31.176: icmp_seq=1 ttl=64 time=0.271 ms
64 bytes from 192.168.31.176: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 192.168.31.176: icmp_seq=3 ttl=64 time=0.226 ms

--- 192.168.31.176 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.226/0.253/0.271/0.019 ms
admin@Attacker:~$
```

Рисунок 29 – Одиночные ICMP-пакеты

```
[2025-06-06 21:53:37][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:53:37][WARNING][heuristic_engine] Heuristic alert: Empty TCP packet from IP 192.168.31.176
[2025-06-06 21:53:37][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 21:54:23][WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 192.168.31.1
[2025-06-06 21:54:23][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 21:56:08][INFO][heuristic_engine] Heuristic alert: ICMP Echo Request from IP 192.168.31.192
"/opt/hybridids/logs/hybridIDS.log" [readonly] 309L, 29254B
```

Рисунок 30 – Отсутствие блокировки одиночных пакетов

Отправка ICMP Echo-запросов с интервалом в 1 мс, что симулирует ICMP flood. Проверяются возможности фильтрации высокочастотных ICMP-запросов, типичных при DDoS-атаках. Представлено на рисунках 31 и 32.

```
len=46 ip=192.168.31.176 ttl=64 id=60359 icmp_seq=12997 rtt=0.3 ms
len=46 ip=192.168.31.176 ttl=64 id=60361 icmp_seq=12998 rtt=0.8 ms
len=46 ip=192.168.31.176 ttl=64 id=60362 icmp_seq=12999 rtt=2.6 ms
len=46 ip=192.168.31.176 ttl=64 id=60363 icmp_seq=13000 rtt=1.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60364 icmp_seq=13001 rtt=2.5 ms
len=46 ip=192.168.31.176 ttl=64 id=60365 icmp_seq=13002 rtt=1.4 ms
len=46 ip=192.168.31.176 ttl=64 id=60366 icmp_seq=13003 rtt=2.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60367 icmp_seq=13004 rtt=1.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60368 icmp_seq=13005 rtt=2.4 ms
len=46 ip=192.168.31.176 ttl=64 id=60369 icmp_seq=13006 rtt=2.7 ms
len=46 ip=192.168.31.176 ttl=64 id=60370 icmp_seq=13007 rtt=1.5 ms
len=46 ip=192.168.31.176 ttl=64 id=60371 icmp_seq=13008 rtt=2.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60372 icmp_seq=13009 rtt=1.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60373 icmp_seq=13010 rtt=3.4 ms
len=46 ip=192.168.31.176 ttl=64 id=60374 icmp_seq=13011 rtt=2.2 ms
len=46 ip=192.168.31.176 ttl=64 id=60376 icmp_seq=13012 rtt=0.9 ms
len=46 ip=192.168.31.176 ttl=64 id=60377 icmp_seq=13013 rtt=1.5 ms
len=46 ip=192.168.31.176 ttl=64 id=60378 icmp_seq=13014 rtt=1.6 ms
len=46 ip=192.168.31.176 ttl=64 id=60379 icmp_seq=13015 rtt=3.1 ms
len=46 ip=192.168.31.176 ttl=64 id=60380 icmp_seq=13016 rtt=1.6 ms
len=46 ip=192.168.31.176 ttl=64 id=60382 icmp_seq=13017 rtt=2.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60383 icmp_seq=13018 rtt=0.5 ms
len=46 ip=192.168.31.176 ttl=64 id=60386 icmp_seq=13020 rtt=1.3 ms
len=46 ip=192.168.31.176 ttl=64 id=60387 icmp_seq=13021 rtt=2.6 ms
len=46 ip=192.168.31.176 ttl=64 id=60388 icmp_seq=13022 rtt=1.1 ms
len=46 ip=192.168.31.176 ttl=64 id=60389 icmp_seq=13023 rtt=2.0 ms
len=46 ip=192.168.31.176 ttl=64 id=60390 icmp_seq=13024 rtt=1.0 ms
```

Рисунок 31 – Спам эхо-запросами

```
[2025-06-06 21:56:08][INFO][heuristic_engine] Heuristic alert: ICMP Echo Request from IP 192.168.31.192
[2025-06-06 21:58:02][INFO][heuristic_engine] Heuristic alert: ICMP Echo Request from IP 192.168.31.192
[2025-06-06 21:58:03][WARNING][heuristic_engine] Heuristic alert: High ICMP packet rate from IP 192.168.31.192
[2025-06-06 21:58:03][INFO][nftables] IP blocking via nftables: 192.168.31.176
"/opt/hybridids/logs/hybridIDS.log" [readonly] 313L, 29659B
```

Рисунок 32 – Блокировка спама

Далее внес IP-адрес своей второй виртуальной машины в правила на блокировку всего TCP-трафика с нее и попробовал отправить пакеты. Изображено на рисунке 33.

```

1 rtt min/avg/max/mdev = 0.204/0.221/0.249/0.011 ms
admin@Attacker:~$ sudo hping3 -S -p 80 --flood 192.168.31.176
HPING 192.168.31.176 (enp0s3 192.168.31.176): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.31.176 hping statistic ---
1448075 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
admin@Attacker:~$

```

Рисунок 33 – Отправка пакетов TCP

Все пакеты были заблокированы сигнатурным движком.

```

[2025-06-06 22:05:18][WARNING][rule_engine] Suspicious package detected: Protocol: TCP, 192.168.31.192:2257 → 192.168.31.176:80
[2025-06-06 22:05:18][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 22:05:18][WARNING][rule_engine] Suspicious package detected: Protocol: TCP, 192.168.31.176:80 → 192.168.31.192:2257
[2025-06-06 22:05:18][INFO][nftables] IP blocking via nftables: 192.168.31.176
[2025-06-06 22:05:28][WARNING][rule_engine] IP already blocked: 192.168.31.192
[2025-06-06 22:05:28][INFO][nftables] IP blocking via nftables: 192.168.31.192
[2025-06-06 22:05:28][WARNING][rule_engine] IP already blocked: 192.168.31.176

```

Рисунок 34 – Блокированный адрес

Далее я внес ВМ, с которой я атакую сервер в белый список. И произвел атаку. Изображено на рисунках 35 и 36.

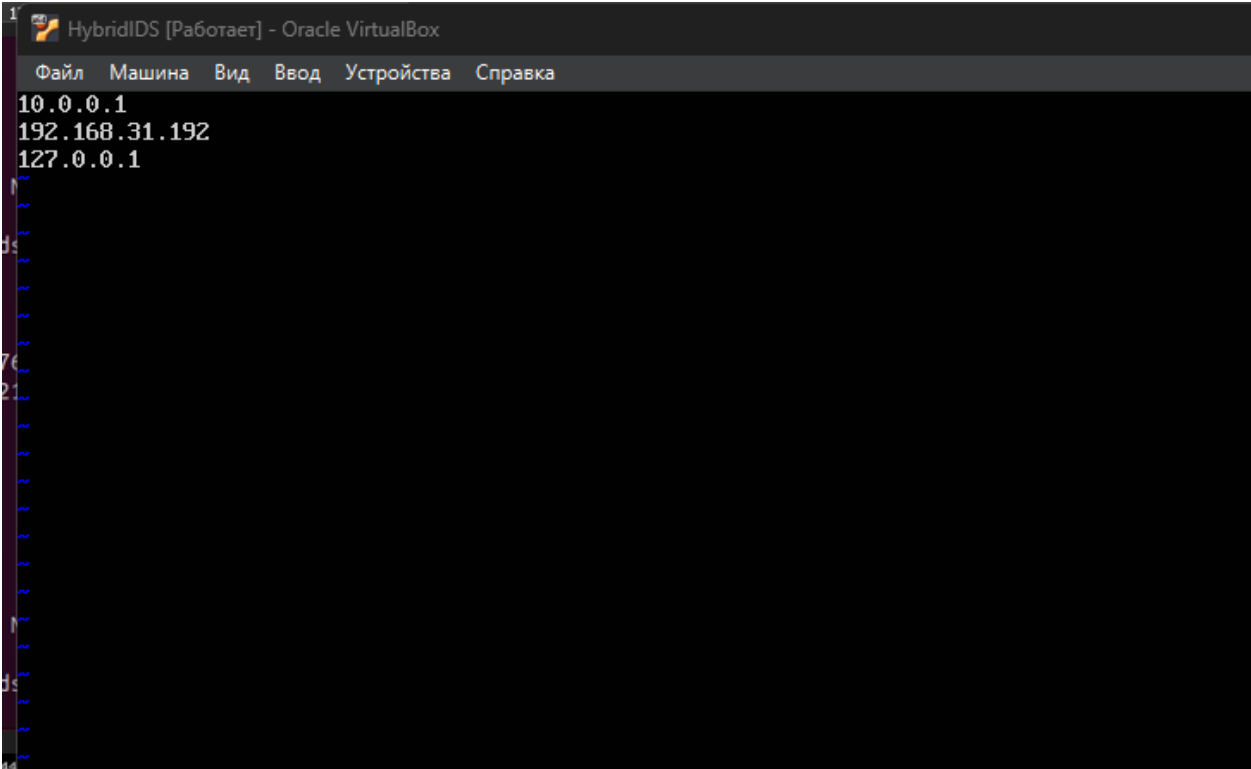


Рисунок 35 – Внесение адреса в белый список

```
admin@Attacker: ~
n=33
SENT (2.0338s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0359s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0382s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0402s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0427s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0447s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0470s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0492s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0512s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0532s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0552s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33
SENT (2.0572s) UDP 192.168.31.192:53 > 192.168.31.176:40125 ttl=64 id=35360 iplen=33
n=33

Max rtt: 0.483ms | Min rtt: 0.122ms | Avg rtt: 0.213ms
Raw packets sent: 1000 (33.000KB) | Rcvd: 8 (488B) | Lost: 992 (99.20%)
Nping done: 1 IP address pinged in 3.07 seconds
admin@Attacker:~$
```

Рисунок 36 – Отправка спам-атаки

Сигнатурный движок распознал, что данный IP-адрес находится в белом списке и не заблокировал его. Изображено на рисунке 37.

```
[2025-06-06 22:07:31][INFO][main] Starting hybridIDS
[2025-06-06 22:07:31][INFO][rule_engine] Rules loaded successfully: 2
[2025-06-06 22:07:31][INFO][rule_engine] Loaded whitelist from file: config/whitelist.txt
[2025-06-06 22:07:55][INFO][rule_engine] IP from the white list, skip: 192.168.31.192
```

Рисунок 37 – Отсутствие реакции

5.2.3 Вывод

Данные тесты направлены не только на выявление массовых атак, но и на обнаружение разведывательной активности – предшественника целевых атак.

Произведенные тесты подтвердили, что система IDS/IPS работает исправно и эффективно выявляет угрозы, в затем быстро нейтрализует их.

6 Охрана труда

Охрана труда в сфере информационных технологий имеет свою специфику и охватывает как физическую безопасность рабочего места, так и цифровую защиту информации и систем, с которыми работает разработчик. При выполнении дипломной работы были соблюдены все актуальные нормативные и рекомендательные требования в данной области.

Организация рабочего места

Работа велась на персональном компьютере, установленном в стандартном офисном помещении. Рабочее место соответствовало требованиям ГОСТ 12.2.032-78 и СанПиН 2.2.2/2.4.1340-03:

- стол и кресло эргономичны, с возможностью регулировки высоты и угла наклона;
- расстояние от глаз до монитора составляло не менее 60 см;
- освещение соответствовало нормативу: освещённость не менее 300 лк при использовании светодиодных светильников с цветовой температурой около 4000 К;
- рабочая зона была оборудована средствами для снижения напряжения зрения (монитор с антибликовым покрытием, режим «тёплого» света во второй половине дня);
- в соответствии с рекомендациями Минздрава РФ, соблюдался режим труда и отдыха;
- ежечасные перерывы по 10–15 минут с выполнением простых упражнений для глаз, кистей рук и позвоночника;
- продолжительность непрерывной работы за компьютером не превышала 4 часов без перерыва.

6.1 Электробезопасность

Разработка и тестирование велись на домашнем оборудовании (персональный компьютер, монитор, периферия), соответствующим классу защиты I. Были соблюдены основные меры электробезопасности:

- все устройства подключались через заземлённые розетки и сетевые фильтры;
- исключалось одновременное касание металлических корпусов устройств при работе;
- проводилось регулярное визуальное осмотры кабелей и разъёмов на предмет износа и повреждений.

Так как проект касался анализа и фильтрации сетевого трафика, особое внимание уделялось программной и информационной безопасности:

- все тесты проводились в изоляции от общедоступных сетей с использованием локальных виртуальных машин;
- разработка и отладка велись в контейнерах Docker, что позволяло минимизировать риски выхода вредоносного трафика в интернет;
- применение Docker также исключало повреждение основной системы при возможных сбоях, перехватах или ошибках в сетевой логике;
- репозитории и данные проекта хранились в закрытом хранилище с двухфакторной аутентификацией;
- использовались современные средства контроля версий (Git) с шифрованием передаваемых данных (SSH)[4].

6.2 Заключение по охране труда

В процессе выполнения дипломной работы были соблюдены требования охраны труда, электробезопасности и цифровой гигиены. Работа велась в условиях, обеспечивающих как физическую безопасность разработчика, так и защиту информационных ресурсов. Использование современных изолированных

сред разработки и тестирования позволило безопасно экспериментировать с сетевыми сценариями, не подвергая риску другие системы или других пользователей. Это позволило эффективно и безопасно реализовать проект.

					ПГУ1.090301.1.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения дипломной работы была разработана гибридная система обнаружения и предотвращения сетевых атак (IDS/IPS), сочетающая в себе как сигнатурные, так и эвристические методы анализа трафика. Система реализована с упором на расширяемость, автоматизацию и практическую применимость в реальных условиях.

В рамках проекта удалось реализовать следующие ключевые компоненты:

- сигнатурный модуль анализа, позволяющий выявлять заранее известные типы атак на основе заданных правил;
- эвристический модуль, анализирующий аномальную сетевую активность;
- автоматизированную сборку и конфигурацию через JSON-файлы;
- логирование с возможностью фильтрации по уровню важности;
- систему автотестирования, обеспечивающую устойчивость и корректность базовых функций;
- подсистему блокировки трафика на основе nftables, интегрированную напрямую с ядром системы.

На текущий момент разработанная система способна:

- вести обнаружение потенциальных вторжений в режиме реального времени;
- автоматически блокировать вредоносные IP-адреса;
- вести подробный лог событий с указанием причины срабатывания и подробной информации о сетевых атрибутах;
- работать в режиме live-захвата с сетевого интерфейса.

Система обладает большим потенциалом для дальнейшего развития. Среди приоритетных направлений можно выделить:

- разработка графического интерфейса для настройки правил, белых списков и мониторинга текущего состояния;

- интеграцию с системами управления информационной безопасностью (SIEM) для централизованного анализа инцидентов;
- расширение эвристических алгоритмов на базе машинного обучения, что позволит системе адаптироваться к новым типам угроз без явных сигнатур;
- масштабирование решения для распределённой работы на нескольких узлах в рамках корпоративной сети.

Разработка гибридной системы обнаружения вторжений была выполнена с учётом принципов экономической эффективности, открытости технологий и дальнейшей масштабируемости решения. Следует также разобрать целесообразность разработки собственной IDS-системы по сравнению с покупкой и внедрением коммерческих аналогов. В таблице 3 я привел данные по затраченным средствам на систему.

Таблица 3 – Расчет затрат на создание системы

Этап работы	Время (часы)	Ставка (руб./час)	Итого (руб.)
Проектирование архитектуры	20	500	10000
Разработка модулей	60	500	30000
Интеграция и отладка	30	500	15000
Документация и тестирование	20	400	8000
Итого:	130		63000

Таким образом, оценочная стоимость полной разработки системы составляет около 60 тысяч рублей.

Использование открытых технологий

В процессе разработки использовались исключительно бесплатные и открытые компоненты, что позволило существенно снизить затраты:

- libpcap – библиотека для захвата сетевых пакетов;
- nlohmann/json – удобный и мощный JSON-парсер;
- Google Test (gtest) – инструмент для модульного тестирования;
- Docker – контейнеризация;
- CMake – свободное средство сборки и компиляции.

Использование этих инструментов исключает расходы на лицензии и позволяет свободно распространять и развивать решение в дальнейшем. С экономической точки зрения, это обеспечивает нулевые эксплуатационные издержки на уровне программной платформы.

Сравнение с коммерческими проектами

Для оценки экономической эффективности рассмотрим стоимость одного из популярных коммерческих решений:

- Snort Enterprise. Подписка для одного узла — от 2500 USD;
- Suricata Pro. Стоимость подписки может составлять от 1000–2000 USD в месяц при полной интеграции с SIEM.

Таким образом, затраты на коммерческое решение превышают в 3–4 раза затраты на разработку собственного аналога. В случае расширения сети или внедрения в учебных заведениях разница будет ещё существеннее.

Разработанная система может быть эффективно использована:

- в образовательных учреждениях для обучения студентов основам сетевой безопасности и работы с трафиком;
- в исследовательских лабораториях для экспериментов с эвристиками и моделями угроз;
- в небольших компаниях и стартапах, где важно минимизировать расходы на ИБ при сохранении контроля за сетью.

Система построена модульно и легко масштабируется:

– возможность запуска в изолированных контейнерах позволяет использовать её на нескольких узлах одновременно;

– простая структура правил и использование стандартных форматов конфигурации (JSON, текстовые списки) делает её удобной в сопровождении;

– отсутствие зависимости от проприетарных библиотек снижает риск «заморозки» проекта.

Поддержка и обновление могут быть реализованы силами одного инженера, без необходимости покупки дополнительных инструментов или лицензий.

Разработка собственной IDS-системы оказалась экономически обоснованной альтернативой коммерческим продуктам. Благодаря применению бесплатных инструментов, небольшой продолжительности разработки и отсутствию затрат на покупку лицензий, проект является доступным, масштабируемым и пригодным для широкого круга задач.

В целом, проделанная работа позволила реализовать гибкую, модульную и эффективную систему обеспечения сетевой безопасности, которая может быть использована как в малых организациях, так и на базе более крупных инфраструктур. Проект закладывает надёжную основу для дальнейших исследований и улучшений в области обнаружения угроз в сети.

Список использованных источников

1. Деннинг Д.Е. Модель обнаружения вторжений // IEEE Transactions on Software Engineering. — 1987. — Vol. SE-13, № 2. — P. 222–232.
2. Roesch M. Snort — Lightweight Intrusion Detection for Networks // Proc. LISA '99: 13th USENIX Conference on System Administration. — 1999.
3. Axelsson S. Intrusion Detection Systems: A Survey and Taxonomy [Электронный ресурс]. — Chalmers University of Technology, 2000. — Режим доступа: <https://www.cs.chalmers.se/~axelsson/papers/ads-survey.pdf> (дата обращения: 01.05.2025).
4. Бишоп М. Компьютерная безопасность: искусство и наука / М. Бишоп. — М.: Вильямс, 2003. — 700 с.
5. Paxson V. Bro: A System for Detecting Network Intruders in Real-Time // Computer Networks. — 1999. — Vol. 31, № 23-24. — P. 2435–2463.
6. nlohmann M. JSON for Modern C++ [Электронный ресурс]. — Режим доступа: <https://github.com/nlohmann/json> (дата обращения: 11.04.2025).
7. Google Test (gtest) Documentation [Электронный ресурс]. — Режим доступа: <https://github.com/google/googletest> (дата обращения: 22.05.2025).
8. Docker Documentation [Электронный ресурс]. — Режим доступа: <https://docs.docker.com/> (дата обращения: 07.05.2025).
9. CMake Documentation [Электронный ресурс]. — Режим доступа: <https://cmake.org/documentation/> (дата обращения: 07.05.2025).
10. Conan — The C/C++ Package Manager [Электронный ресурс]. — Режим доступа: <https://conan.io/> (дата обращения: 15.05.2025).
11. Paxson V. Bro: A System for Detecting Network Intruders in Real-Time // Proceedings of the 7th USENIX Security Symposium. — 1998.

Приложение А

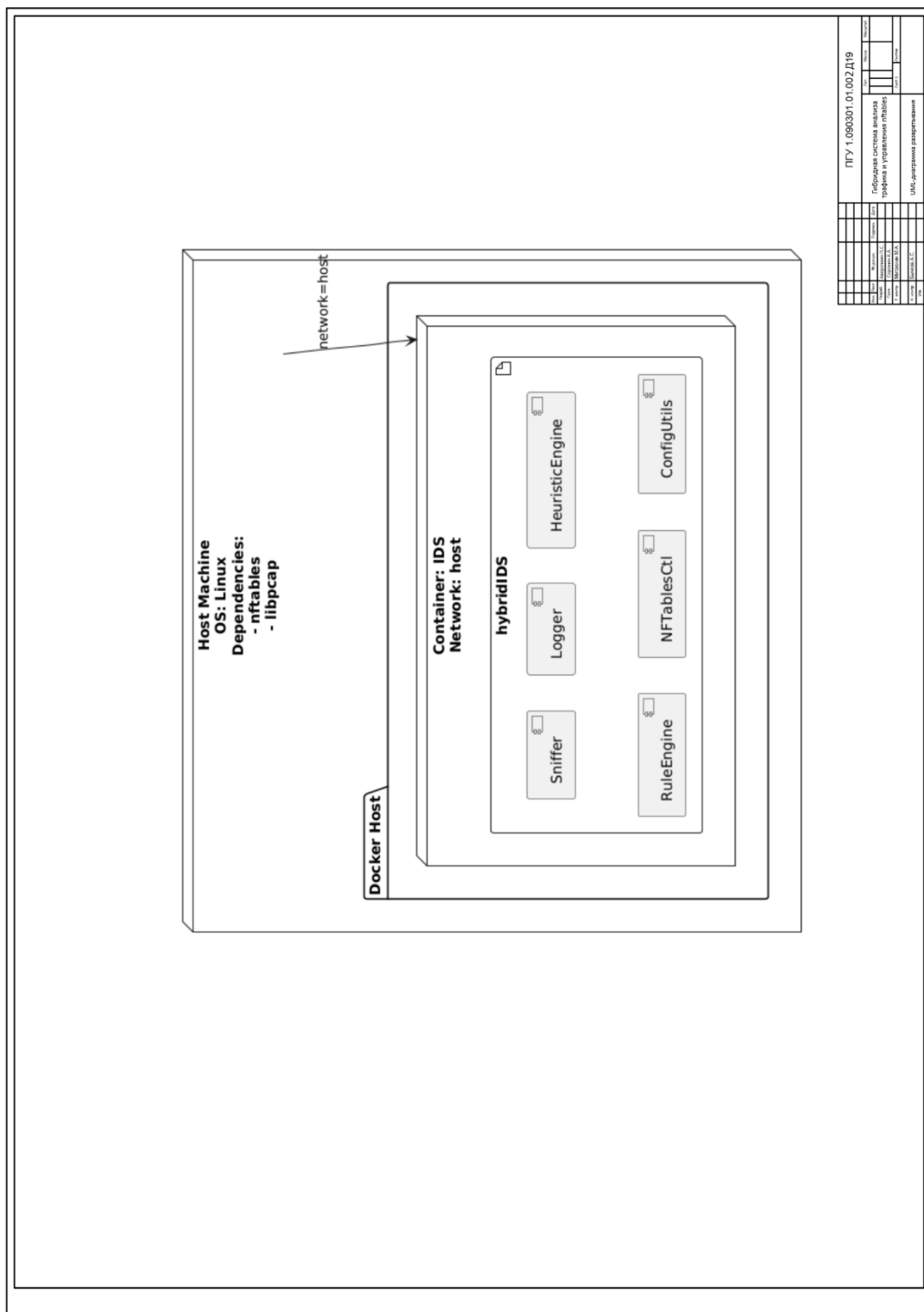
(Обязательное)

Гибридная система анализа трафика и управления nftables

UML – диаграммы

					<i>ПГУ1.090301.1.001 ПЗ</i>	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма развертывания



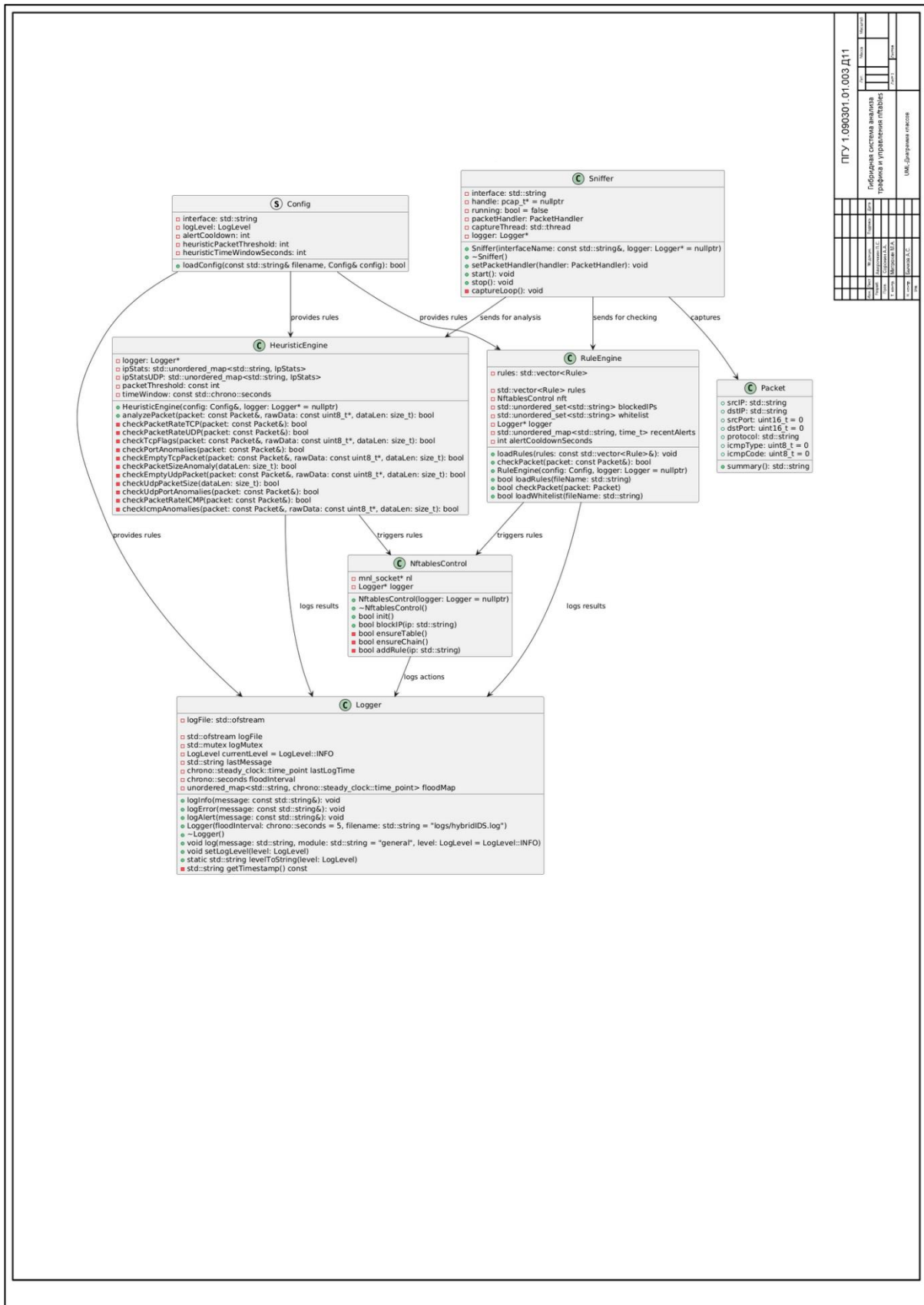
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.1.001 ПЗ

Лист

77

Приложение А.2-Диаграмма классов



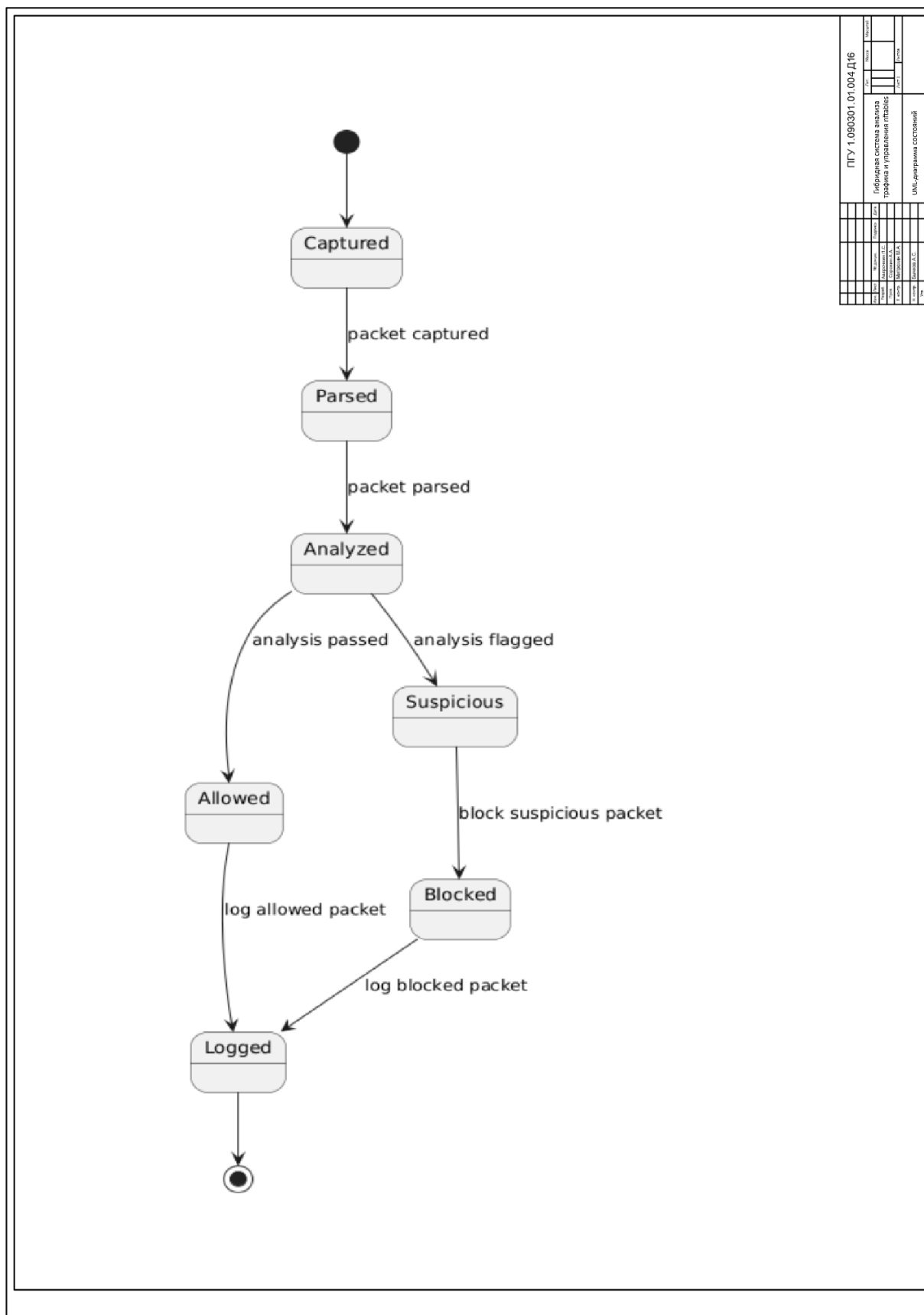
ПГУ 1.090301.01.003 Д11	
Изм.	Лист
№ докум.	Подпись
Дата	Дата

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ1.090301.1.001 ПЗ

Лист
78

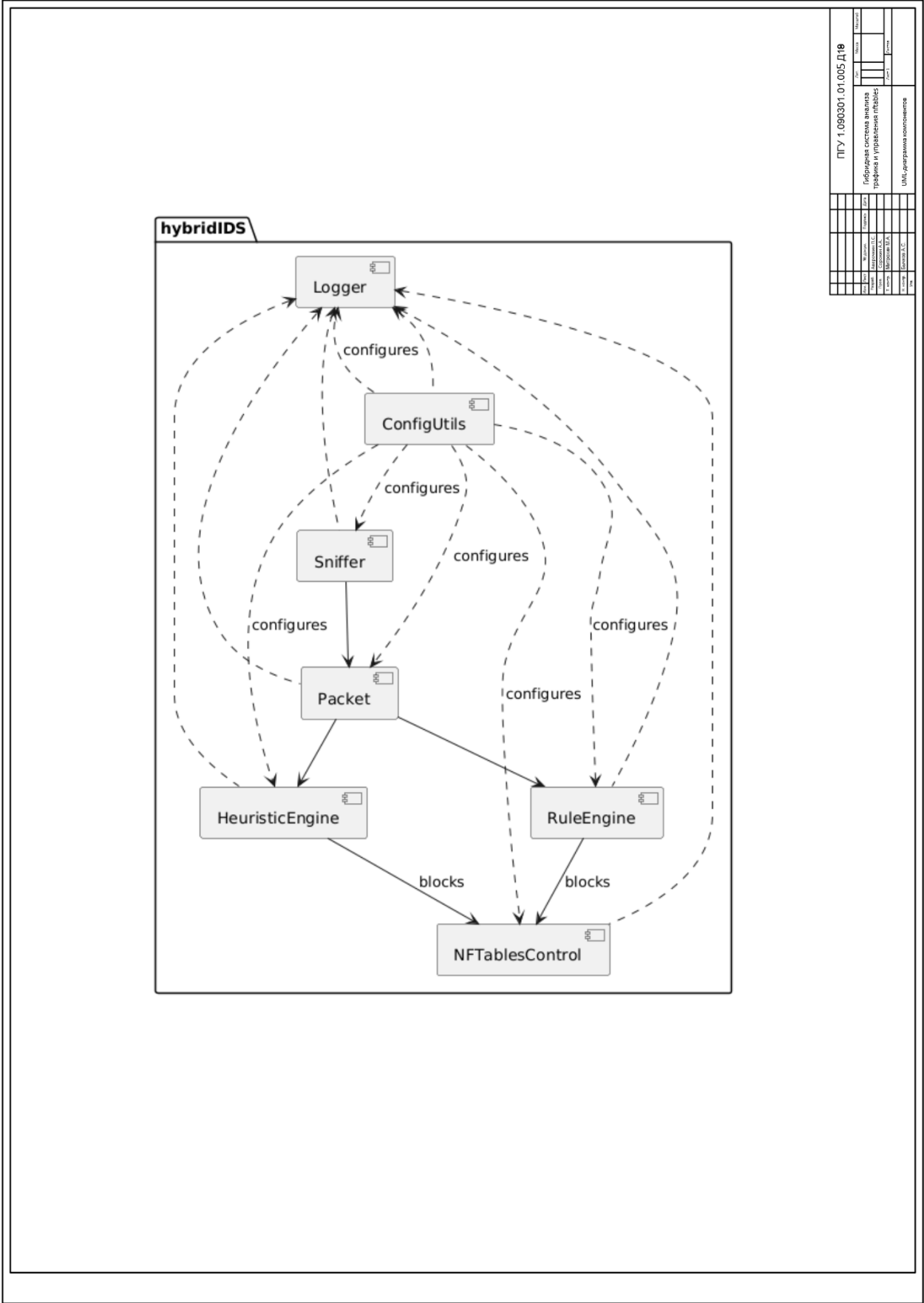
Приложение А.3-Диаграмма состояний



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.1.001 ПЗ

Приложение А.4-Диаграмма компонентов



Приложение Б

(Обязательно)

Гибридная система анализа трафика и управления nftables

Листинг программы

					ПГУ1.090301.1.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1-Листинг основного модуля (main.cpp)

В данном приложении представлен основной файл сервиса и его КОМПОНЕНТЫ.

```
#include <iostream>
#include <csignal>
#include <atomic>
#include <thread>
#include <chrono>

#include "sniffer.h"
#include "rule_engine.h"
#include "nftables_control.h"
#include "logger.h"
#include "packet.h"
#include "config_utils.h"
#include "heuristic_engine.h"

std::string configPath = "config/config.json";

std::atomic<bool> running(true);
Logger* globalLogger = nullptr;

void signalHandler(int signum)
{
    if (globalLogger)
        globalLogger->log("Termination signal received (" +
std::to_string(signum) + ")", "main", LogLevel::INFO);
    running = false;
}

int main()
{
    Config config;
    if (!loadConfig(configPath, config))
    {
        std::cerr << "Failed to load configuration from
config.json\n";
        return 1;
    }
    Logger logger( std::chrono::seconds(config.alertCooldown) );
    logger.setLogLevel(config.logLevel);
    globalLogger = &logger;

    std::signal(SIGINT, signalHandler);
    std::signal(SIGTERM, signalHandler);

    logger.log(std::string("Logging level: ") +
Logger::levelToString(config.logLevel), "main", LogLevel::INFO);
```

```

    logger.log("Interface: " + config.interface, "main",
LogLevel::INFO);
    logger.log("Starting hybridIDS", "main", LogLevel::INFO);

    RuleEngine ruleEngine(config, &logger);
    if (!ruleEngine.loadRules("config/rules.json"))
    {
        logger.log("Error loading rules", "rule_engine",
LogLevel::ERROR);
        return 1;
    }

    if (!ruleEngine.loadWhitelist("config/whitelist.txt"))
    {
        logger.log("Error loading whitelist", "rule_engine",
LogLevel::ERROR);
        return 1;
    }

    HeuristicEngine heuristicEngine(config, &logger);
    NftablesControl nftables(&logger);
    Sniffer sniffer(config.interface, &logger);

    sniffer.setPacketHandler([&](const Packet& pkt, const
uint8_t* rawData, size_t dataLen)
    {
        bool sigDetect = ruleEngine.checkPacket(pkt);
        bool heurDetect = heuristicEngine.analyzePacket(pkt,
rawData, dataLen);
        if (sigDetect || heurDetect)
        {
            logger.log("Threat detected from IP: " + pkt.srcIP,
"main", LogLevel::WARNING);
            nftables.blockIP(pkt.srcIP);
        }
    });

    sniffer.start();

    while (running)
    {
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    logger.log("Stop hybridIDS", "main", LogLevel::INFO);
    sniffer.stop();

    logger.log("hybridIDS completed", "main", LogLevel::INFO);
    return 0;
}

```

Приложение Б.2-Листинг заголовочного файла модуля перехвата трафика (sniffer.h)

В данном приложении представлен файл определяющий интерфейс. класса Sniffer.

```
#pragma once

#include <functional>
#include <string>
#include <thread>
#include <pcap.h>

#include "packet.h"
#include "logger.h"

class Sniffer
{
public:
    using PacketHandler = std::function<void(const Packet&,
const uint8_t*, size_t)>;

    Sniffer(const std::string& interfaceName, Logger* logger =
nullptr);
    ~Sniffer();

    void setPacketHandler(PacketHandler handler);

    void start();
    void stop();

private:
    std::string interface;
    pcap_t* handle = nullptr;
    bool running = false;
    PacketHandler packetHandler;
    std::thread captureThread;

    Logger* logger;

    void captureLoop();
};
```

Приложение Б.3-Листинг модуля перехвата трафика (sniffer.cpp)

В данном приложении представлен файл, содержащий реализацию модуля захвата сетевых пакетов с использованием библиотеки libpcap.

```
#include "sniffer.h"

#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <net/ethernet.h> // для ether_header
#include <arpa/inet.h>

Sniffer::Sniffer(const std::string& interfaceName, Logger*
logger) : interface(interfaceName), logger(logger){}

Sniffer::~Sniffer()
{
    stop();
    if (handle)
    {
        pcap_close(handle);
        handle = nullptr;
    }
}

void Sniffer::setPacketHandler(PacketHandler handler)
{
    packetHandler = handler;
}

void Sniffer::start()
{
    char errbuf[PCAP_ERRBUF_SIZE];
    handle = pcap_open_live(interface.c_str(), BUFSIZ, 1, 1000,
errbuf);
    if (!handle)
    {
        if (logger)
            logger->log(std::string("Error opening interface ")
+ interface + ": " + errbuf, "sniffer", LogLevel::ERROR);
        return;
    }

    running = true;
    captureThread = std::thread([this]() { captureLoop(); });
}

void Sniffer::stop()
```

```

{
    running = false;
    if (handle)
    {
        pcap_breakloop(handle);
    }
    if (captureThread.joinable())
        captureThread.join();
}

void Sniffer::captureLoop()
{
    while (running)
    {
        struct pcap_pkthdr* header;
        const u_char* data;
        int result = pcap_next_ex(handle, &header, &data);
        if (result == 1)
        {
            if (header->caplen < sizeof(ether_header))
                continue; // Мало данных

            Packet pkt;

            // Разбор Ethernet
            const struct ether_header* eth = (const struct
ether_header*)data;
            if (ntohs(eth->ether_type) != ETHERTYPE_IP)
                continue; // Только IPv4

            if (header->caplen < sizeof(ether_header) +
sizeof(struct ip))
                continue; // Мало данных для IP

            // Разбор IP
            const struct ip* iphdr = (const struct ip*)(data +
sizeof(struct ether_header));
            pkt.srcIP = inet_ntoa(iphdr->ip_src);
            pkt.dstIP = inet_ntoa(iphdr->ip_dst);

            const u_char* transportHeader = data +
sizeof(ether_header) + iphdr->ip_hl * 4;

            // Протокол
            switch (iphdr->ip_p)
            {
                case IPPROTO_TCP:
                {
                    pkt.protocol = "TCP";
                }
            }
        }
    }
}

```

```

        if (header->caplen < sizeof(ether_header) +
            iphdr->ip_hl * 4 + sizeof(tcphdr))
            continue; // Мало данных для TCP

        const struct tcphdr* tcph = (const struct
tcphdr*) (data + sizeof(ether_header) + iphdr->ip_hl * 4);
        pkt.srcPort = ntohs(tcph->th_sport);
        pkt.dstPort = ntohs(tcph->th_dport);
        break;
    }
    case IPPROTO_UDP:
    {
        pkt.protocol = "UDP";

        if (header->caplen < sizeof(ether_header) +
            iphdr->ip_hl * 4 + sizeof(udphdr))
            continue; // Мало данных для UDP

        const struct udphdr* udph = (const struct
udphdr*) (data + sizeof(ether_header) + iphdr->ip_hl * 4);
        pkt.srcPort = ntohs(udph->uh_sport);
        pkt.dstPort = ntohs(udph->uh_dport);
        break;
    }
    case IPPROTO_ICMP:
    {
        pkt.protocol = "ICMP";
        const uint8_t* icmpHeader = transportHeader;
        pkt.icmpType = icmpHeader[0];
        pkt.icmpCode = icmpHeader[1];
        break;
    }
    default:
        pkt.protocol = "OTHER";
        pkt.srcPort = 0;
        pkt.dstPort = 0;
        break;
    }

    if (packetHandler)
        packetHandler(pkt, data, header->caplen);
}
else if (result == -1)
{
    if (logger)
        logger->log(std::string("pcap_next_ex error: ")
+ pcap_geterr(handle), "sniffer", LogLevel::DEBUG);
    break;
}
}
}

```

Приложение Б.4-Листинг заголовочного файла модуля анализа правил (rule_engine.h)

В данном приложении объявлен класс, обеспечивающий анализ сетевых пакетов на основе сигнатурных правил.

```
#pragma once

#include <vector>
#include <string>
#include <unordered_set>

#include "logger.h"
#include "packet.h"
#include "nftables_control.h"
#include "config_utils.h"

struct Rule
{
    std::string srcIP;
    std::string dstIP;
    int srcPort = -1;
    int dstPort = -1;
    std::string protocol;

    int icmpType = -1;
    int icmpCode = -1;
};

class RuleEngine
{
public:
    RuleEngine(const Config& config, Logger* logger = nullptr);

    bool loadRules(const std::string& fileName);
    bool checkPacket(const Packet& packet);
    bool loadWhitelist(const std::string& fileName);

private:
    std::vector<Rule> rules;
    NftablesControl nft;
    std::unordered_set<std::string> blockedIPs;
    std::unordered_set<std::string> whitelist;
    Logger* logger;

    std::unordered_map<std::string, time_t> recentAlerts;
    int alertCooldownSeconds;
};
```

Приложение Б.5-Листинг файла модуля анализа правил (rule_engine.cpp)

В данном приложении представлена реализация модуля, обеспечивающего анализ сетевых пакетов на основе сигнатурных правил.

```
#include <fstream>
#include <iostream>
#include <nlohmann/json.hpp>

#include "rule_engine.h"

using json = nlohmann::json;

RuleEngine::RuleEngine(const Config& config, Logger* logger) :
logger(logger), alertCooldownSeconds(config.alertCooldown)
{
    if (!nft.init())
    {
        if(logger)
            logger->log("Error initializing NftablesControl",
"rule_engine", LogLevel::ERROR);
    }
}

bool RuleEngine::loadRules(const std::string& fileName)
{
    std::ifstream file(fileName);
    if (!file.is_open())
    {
        if (logger)
            logger->log("Failed to open rules file: " +
fileName, "rule_engine", LogLevel::ERROR);
        return false;
    }

    json j;
    try
    {
        file >> j;
    }
    catch (const json::parse_error& e)
    {
        if (logger)
            logger->log(std::string("Error parsing JSON rules:
") + e.what(), "rule_engine", LogLevel::ERROR);
        return false;
    }
    if (!j.is_array())
    {
        if(logger)
```

```

        logger->log("The format of the rules file must be a
JSON array", "rule_engine", LogLevel::ERROR);
        return false;
    }
    rules.clear();

    for (const auto& item : j)
    {
        Rule rule;

        if(item.contains("src_ip"))
            rule.srcIP = item["src_ip"].get<std::string>();
        if(item.contains("dst_ip"))
            rule.dstIP = item["dst_ip"].get<std::string>();
        if(item.contains("src_port"))
            rule.srcPort = item["src_port"].get<int>();
        if(item.contains("dst_port"))
            rule.dstPort = item["dst_port"].get<int>();
        if(item.contains("protocol"))
            rule.protocol = item["protocol"].get<std::string>();

        if (rule.srcPort < -1 || rule.srcPort > 65535)
        {
            if (logger)
                logger->log("Incorrect srcPort in rule: " +
std::to_string(rule.srcPort), "rule_engine", LogLevel::DEBUG);
            continue;
        }
        if (rule.dstPort < -1 || rule.dstPort > 65535)
        {
            if(logger)
                logger->log("Incorrect dstPort in rule: " +
std::to_string(rule.dstPort), "rule_engine", LogLevel::DEBUG);
            continue;
        }

        rules.push_back(rule);
    }
    if(logger)
        logger->log("Rules loaded successfully: " +
std::to_string(rules.size()), "rule_engine", LogLevel::INFO);
    return true;
}

bool RuleEngine::checkPacket(const Packet& packet)
{
    if (whitelist.contains(packet.srcIP))
    {
        if(logger)

```

```

        logger->log("IP from the white list, skip: " +
packet.srcIP, "rule_engine", LogLevel::INFO);
        return false;
    }

    for (const auto& rule : rules)
    {
        if (!rule.srcIP.empty() && rule.srcIP != packet.srcIP)
            continue;
        if (!rule.dstIP.empty() && rule.dstIP != packet.dstIP)
            continue;
        if (rule.srcPort != -1 && rule.srcPort !=
packet.srcPort)
            continue;
        if (rule.dstPort != -1 && rule.dstPort !=
packet.dstPort)
            continue;
        if (!rule.protocol.empty() && rule.protocol !=
packet.protocol)
            continue;

        time_t now = std::time(nullptr);
        auto it = recentAlerts.find(packet.srcIP);
        if (it == recentAlerts.end() || difftime(now, it-
>second) > alertCooldownSeconds)
        {
            recentAlerts[packet.srcIP] = now;
            if(logger)
                logger->log("Suspicious package detected: " +
packet.summary(), "rule_engine", LogLevel::WARNING);
        }
        // Блокируем IP
        if (!blockedIPs.contains(packet.srcIP))
        {
            if (nft.blockIP(packet.srcIP))
            {
                this->blockedIPs.insert(packet.srcIP);
                if(logger)
                    logger->log(std::string("IP blocked by rule:
") + packet.srcIP, "rule_engine", LogLevel::WARNING);
            }
            else
            {
                if(logger)
                    logger->log(std::string("IP blocking error:
") + packet.srcIP, "rule_engine", LogLevel::ERROR);
            }
        }
        else
        {
            if(logger)

```

```

        logger->log(std::string("IP already blocked: ") +
packet.srcIP, "rule_engine", LogLevel::WARNING);
    }
    return true;
}

return false;
}

bool RuleEngine::loadWhitelist(const std::string& fileName)
{
    std::ifstream file(fileName);
    if (!file.is_open())
    {
        if(logger)
            logger->log(std::string("Failed to open file
whitelist: ") + fileName, "rule_engine", LogLevel::ERROR);
        return false;
    }

    std::string line;
    while (std::getline(file, line))
    {
        if (!line.empty())
            this->whitelist.insert(line);
    }

    if(logger)
        logger->log(std::string("Loaded whitelist from file: ")
+ fileName, "rule_engine", LogLevel::INFO);
    return true;
}

```

Приложение Б.6-Листинг заголовочного файла модуля управления блокировкой (nftables_control.h)

В данном приложении объявлены класс и методы взаимодействия с nftables для блокировки IP-адресов.

```
#pragma once

#include <string>
#include <libmnl/libmnl.h>

#include "logger.h"

class NftablesControl
{
public:
    NftablesControl(Logger* logger = nullptr);
    ~NftablesControl();

    bool init();
    bool blockIP(const std::string& ip);

private:
    struct mnl_socket* nl = nullptr;
    Logger* logger;

    bool ensureTable();
    bool ensureChain();
    bool addRule(const std::string& ip);
};
```

Приложение Б.7-Листинг файла модуля управления блокировкой (nftables_control.cpp)

В данном приложении представлена реализация модуля взаимодействия с nftables для блокировки IP-адресов.

```
#include <arpa/inet.h>

#include <linux/netfilter.h>
#include <linux/netfilter/nf_tables.h>

#include <libnftnl/table.h>
#include <libnftnl/chain.h>
#include <libnftnl/rule.h>
#include <libnftnl/set.h>
#include <libnftnl/expr.h>

#include "nftables_control.h"

NftablesControl::NftablesControl(Logger* logger) : nl(nullptr),
logger(logger)
{
    init();
}

NftablesControl::~NftablesControl()
{
    if (nl != nullptr)
    {
        mnl_socket_close(nl);
        nl = nullptr;
    }
}

bool NftablesControl::init()
{
    if (nl != nullptr)
    {
        // Уже инициализирован
        return true;
    }

    nl = mnl_socket_open(NETLINK_NETFILTER);
    if (nl == nullptr)
    {
        if (logger)
            logger->log("Error opening Netlink socket.",
            "nftables", LogLevel::ERROR);
        return false;
    }
}
```

```

    }

    if (mnl_socket_bind(nl, 0, MNL_SOCKET_AUTOPIID) < 0)
    {
        if (logger)
            logger->log("Error while binding Netlink socket.",
"nftables", LogLevel::ERROR);
        mnl_socket_close(nl);
        nl = nullptr;
        return false;
    }

    if (!ensureTable() || !ensureChain())
    {
        if (logger)
            logger->log("Error creating nftables table or
chain.", "nftables", LogLevel::ERROR);
        return false;
    }

    return true;
}

bool NftablesControl::ensureTable()
{
    struct nftnl_table* table = nftnl_table_alloc();
    if (!table) return false;

    nftnl_table_set_str(table, NFTNL_TABLE_NAME, "filter");
    nftnl_table_set_u32(table, NFTNL_TABLE_FAMILY,
NFPROTO_INET);

    char buf[MNL_SOCKET_BUFFER_SIZE];
    struct nlmsg_hdr* nlh = nftnl_table_nlmsg_build_hdr(buf,
NFT_MSG_NEWTABLE,
        NFPROTO_INET, NLM_F_CREATE | NLM_F_ACK, 0);

    nftnl_table_nlmsg_build_payload(nlh, table);

    int ret = mnl_socket_sendto(nl, nlh, nlh->nlmsg_len);
    nftnl_table_free(table);

    if (ret < 0) return false;

    ret = mnl_socket_recvfrom(nl, buf, sizeof(buf));
    return (ret >= 0);
}

bool NftablesControl::ensureChain()
{
    struct nftnl_chain* chain = nftnl_chain_alloc();

```

```

    if (!chain) return false;

    nftnl_chain_set_str(chain, NFTNL_CHAIN_TABLE, "filter");
    nftnl_chain_set_str(chain, NFTNL_CHAIN_NAME, "input");
    nftnl_chain_set_u32(chain, NFTNL_CHAIN_FAMILY,
NFPROTO_INET);

    // Вот это – КОРРЕКТНОЕ задание hook'a
    nftnl_chain_set_u32(chain, NFTNL_CHAIN_HOOKNUM,
NF_INET_LOCAL_IN); // "input" по смыслу
    nftnl_chain_set_u32(chain, NFTNL_CHAIN_PRIO, 0);
    nftnl_chain_set_str(chain, NFTNL_CHAIN_TYPE, "filter");

    // Опционально: политика
    nftnl_chain_set_u32(chain, NFTNL_CHAIN_POLICY, NF_ACCEPT);

    char buf[MNL_SOCKET_BUFFER_SIZE];
    struct nlmsg_hdr* nlh = nftnl_chain_nlmsg_build_hdr(buf,
NFT_MSG_NEWCHAIN,
        NFPROTO_INET, NLM_F_CREATE | NLM_F_ACK, 0);

    nftnl_chain_nlmsg_build_payload(nlh, chain);
    int ret = mnl_socket_sendto(nl, nlh, nlh->nlmsg_len);
    nftnl_chain_free(chain);

    if (ret < 0) return false;

    ret = mnl_socket_recvfrom(nl, buf, sizeof(buf));
    return (ret >= 0);
}

```

```

bool NftablesControl::blockIP(const std::string& ip)
{
    if (logger)
        logger->log("IP blocking via nftables: " + ip,
"nftables", LogLevel::INFO);

    struct nftnl_rule* rule = nftnl_rule_alloc();
    if (!rule) return false;

    nftnl_rule_set_str(rule, NFTNL_RULE_TABLE, "filter");
    nftnl_rule_set_str(rule, NFTNL_RULE_CHAIN, "input");
    nftnl_rule_set_u32(rule, NFTNL_RULE_FAMILY, NFPROTO_INET);

    struct nftnl_expr* expr = nftnl_expr_alloc("payload");
    nftnl_expr_set_u32(expr, NFTNL_EXPR_PAYLOAD_BASE,
NFT_PAYLOAD_NETWORK_HEADER);
    nftnl_expr_set_u32(expr, NFTNL_EXPR_PAYLOAD_OFFSET, 12);
    nftnl_expr_set_u32(expr, NFTNL_EXPR_PAYLOAD_LEN, 4);
    nftnl_rule_add_expr(rule, expr);
}

```

```

    expr = nftnl_expr_alloc("cmp");
    nftnl_expr_set_u32(expr, NFTNL_EXPR_CMP_SREG, NFT_REG_1);
    nftnl_expr_set_u32(expr, NFTNL_EXPR_CMP_OP, NFT_CMP_EQ);

    uint32_t ip_bin = inet_addr(ip.c_str());
    nftnl_expr_set_data(expr, NFTNL_EXPR_CMP_DATA, &ip_bin,
sizeof(ip_bin));
    nftnl_rule_add_expr(rule, expr);

    expr = nftnl_expr_alloc("immediate");
    nftnl_expr_set_u32(expr, NFTNL_EXPR_IMM_DREG,
NFT_REG_VERDICT);
    nftnl_expr_set_u32(expr, NFTNL_EXPR_IMM_VERDICT, NF_DROP);
    nftnl_rule_add_expr(rule, expr);

    char buf[MNL_SOCKET_BUFFER_SIZE];
    struct nlmsg_hdr* nlh = nftnl_rule_nlmsg_build_hdr(buf,
NFT_MSG_NEWRULE,
        NFPROTO_INET, NLM_F_CREATE | NLM_F_ACK, 0);

    nftnl_rule_nlmsg_build_payload(nlh, rule);

    int ret = mnl_socket_sendto(nl, nlh, nlh->nlmsg_len);
    nftnl_rule_free(rule);

    if (ret < 0) return false;

    ret = mnl_socket_recvfrom(nl, buf, sizeof(buf));
    return (ret >= 0);
    //return true; // или false в случае ошибки
}

```

Приложение Б.8-Листинг заголовочного файла модуля эвристического анализа трафика (heuristic_engine.h)

В данном приложении объявлены класс и методы позволяющие проводить эвристический анализ сетевых пакетов.

```
#pragma once

#include <string>
#include <unordered_map>
#include <chrono>

#include "packet.h"
#include "logger.h"
#include "config_utils.h"

class HeuristicEngine
{
public:
    HeuristicEngine(Config& config, Logger* logger = nullptr);

    // Возвращает true, если обнаружена подозрительная
    активность
    bool analyzePacket(const Packet& packet, const uint8_t*
rawData, size_t dataLen);

private:
    Logger* logger;

    struct IpStats
    {
        int packetCount = 0;
        std::chrono::steady_clock::time_point firstPacketTime;
    };

    std::unordered_map<std::string, IpStats> ipStats;
    std::unordered_map<std::string, IpStats> ipStatsUDP;

    const int packetThreshold; // Порог пакетов
    const std::chrono::seconds timeWindow; // Временной интервал

    bool checkPacketRateTCP(const Packet& packet);
    bool checkPacketRateUDP(const Packet& packet);

    bool checkTcpFlags(const Packet& packet, const uint8_t*
rawData, size_t dataLen);
    bool checkPortAnomalies(const Packet& packet);
    bool checkEmptyTcpPacket(const Packet& packet, const
uint8_t* rawData, size_t dataLen);
```

```

    bool checkPacketSizeAnomaly(size_t dataLen);

    bool checkEmptyUdpPacket(const Packet& packet, const
uint8_t* rawData, size_t dataLen);
    bool checkUdpPacketSize(size_t dataLen);
    bool checkUdpPortAnomalies(const Packet& packet);

    bool checkPacketRateICMP(const Packet& packet);
    bool checkIcmpAnomalies(const Packet& packet, const uint8_t*
rawData, size_t dataLen);

};

```

					ПГУ1.090301.1.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.9-Листинг файла модуля эвристического анализа трафика (heuristic_engine.cpp)

В данном приложении представлена реализация модуля позволяющего проводить эвристический анализ сетевых пакетов.

```
#include <netinet/tcp.h>
#include <chrono>
#include <iostream>
#include <net/ethernet.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

#include "heuristic_engine.h"

HeuristicEngine::HeuristicEngine(Config& config, Logger* logger)
    : logger(logger),
    packetThreshold(config.heuristicPacketThreshold),
    timeWindow(std::chrono::seconds(config.heuristicTimeWindowSeconds))
{}

bool HeuristicEngine::analyzePacket(const Packet& packet, const
uint8_t* rawData, size_t dataLen)
{
    bool suspicious = false;

    if (packet.protocol == "TCP")
    {
        if (checkPacketRateTCP(packet)) suspicious = true;
        if (checkTcpFlags(packet, rawData, dataLen)) suspicious
= true;
        if (checkEmptyTcpPacket(packet, rawData, dataLen))
suspicious = true;
    }
    else if (packet.protocol == "UDP")
    {
        if (checkPacketRateUDP(packet)) suspicious = true;
        if (checkEmptyUdpPacket(packet, rawData, dataLen))
suspicious = true;
        if (checkUdpPacketSize(dataLen)) suspicious = true;
    }
    if (packet.protocol == "ICMP") {
        if (checkPacketRateICMP(packet)) suspicious = true;
        if (checkIcmpAnomalies(packet, rawData, dataLen))
suspicious = true;
    }

    if (checkPortAnomalies(packet)) suspicious = true;
```

```

        if (checkPacketSizeAnomaly(dataLen)) suspicious = true;

        if (packet.protocol == "UDP")
        {
            if (checkUdpPortAnomalies(packet)) suspicious = true;
        }

        return suspicious;
    }

bool HeuristicEngine::checkPacketRateTCP(const Packet& packet)
{
    using clock = std::chrono::steady_clock;
    auto now = clock::now();

    auto& stats = ipStats[packet.srcIP];
    if (stats.packetCount == 0)
    {
        stats.firstPacketTime = now;
        stats.packetCount = 1;
        return false;
    }

    auto duration =
std::chrono::duration_cast<std::chrono::seconds>(now -
stats.firstPacketTime);
    if (duration > timeWindow)
    {
        stats.packetCount = 1;
        stats.firstPacketTime = now;
        return false;
    }
    else
    {
        stats.packetCount++;
        if (stats.packetCount > packetThreshold)
        {
            if (logger)
                logger->log("Heuristic alert: High packet rate
from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
            stats.packetCount = 0;
            return true;
        }
    }
    return false;
}

bool HeuristicEngine::checkPacketRateUDP(const Packet& packet)
{

```

```

using clock = std::chrono::steady_clock;
auto now = clock::now();

auto& stats = ipStatsUDP[packet.srcIP];
if (stats.packetCount == 0)
{
    stats.firstPacketTime = now;
    stats.packetCount = 1;
    return false;
}

auto duration =
std::chrono::duration_cast<std::chrono::seconds>(now -
stats.firstPacketTime);
if (duration > timeWindow)
{
    stats.packetCount = 1;
    stats.firstPacketTime = now;
    return false;
}
else
{
    stats.packetCount++;
    if (stats.packetCount > packetThreshold)
    {
        if (logger)
            logger->log("Heuristic alert: High UDP packet
rate from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
        stats.packetCount = 0;
        return true;
    }
}
return false;
}

bool HeuristicEngine::checkEmptyUdpPacket(const Packet& packet,
const uint8_t* rawData, size_t dataLen)
{
    if (packet.protocol != "UDP")
        return false;

    if (dataLen < sizeof(ether_header) + sizeof(ip) +
sizeof(udphdr))
        return false;

    const ip* iphdr = (const ip*)(rawData +
sizeof(ether_header));
    size_t ipHeaderLen = iphdr->ip_hl * 4;

```

```

        if (dataLen < sizeof(ether_header) + ipHeaderLen +
sizeof(udphdr))
            return false;

        const udphdr* udph = (const udphdr*)(rawData +
sizeof(ether_header) + ipHeaderLen);

        size_t udpPayloadLen = ntohs(udph->len) - sizeof(udphdr);

        if (udpPayloadLen == 0)
        {
            if (logger)
                logger->log("Heuristic alert: Empty UDP packet from
IP " + packet.srcIP, "heuristic_engine", LogLevel::WARNING);
            return true;
        }
        return false;
    }

bool HeuristicEngine::checkUdpPacketSize(size_t dataLen)
{
    // UDP заголовок – 8 байт, минимальный IP заголовок 20 байт,
Ethernet 14 байт
    // Обычно payload UDP > 8 байт, слишком маленькие и слишком
большие – подозрительно
    const size_t minSize = sizeof(ether_header) + sizeof(ip) +
sizeof(udphdr) + 8; // 8 – минимальный полезный размер
    const size_t maxSize = 576; // Часто используют max UDP
packet size ~576 байт (можно изменить)

    if (dataLen < minSize)
    {
        if (logger)
            logger->log("Heuristic alert: Very small UDP packet
size: " + std::to_string(dataLen), "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    if (dataLen > maxSize)
    {
        if (logger)
            logger->log("Heuristic alert: Very large UDP packet
size: " + std::to_string(dataLen), "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    return false;
}

bool HeuristicEngine::checkUdpPortAnomalies(const Packet&
packet)

```

```

{
    // Порты вне диапазона 1-65535
    if (packet.srcPort < 0 || packet.srcPort > 65535)
    {
        if (logger)
            logger->log("Heuristic alert: Invalid UDP source
port " + std::to_string(packet.srcPort) + " from IP " +
packet.srcIP, "heuristic_engine", LogLevel::WARNING);
        return true;
    }
    if (packet.dstPort < 0 || packet.dstPort > 65535)
    {
        if (logger)
            logger->log("Heuristic alert: Invalid UDP
destination port " + std::to_string(packet.dstPort) + " from IP
" + packet.srcIP, "heuristic_engine", LogLevel::WARNING);
        return true;
    }

    // Много трафика на системные порты (например, 0-1023) без
явной причины подозрительно
    if ((packet.srcPort > 0 && packet.srcPort < 1024) ||
(packet.dstPort > 0 && packet.dstPort < 1024))
    {
        if (logger)
            logger->log("Heuristic alert: UDP traffic on system
port from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
    }

    return false;
}

bool HeuristicEngine::checkTcpFlags(const Packet& packet, const
uint8_t* rawData, size_t dataLen)
{
    // Проверяем TCP флаги на аномалии
    // Данные rawData: ethernet + IP + TCP
    if (dataLen < sizeof(ether_header) + sizeof(ip) +
sizeof(tcphdr))
        return false;

    const ip* iphdr = (const ip*)(rawData +
sizeof(ether_header));
    size_t ipHeaderLen = iphdr->ip_hl * 4;

    if (dataLen < sizeof(ether_header) + ipHeaderLen +
sizeof(tcphdr))
        return false;
}

```

```

    const tcphdr* tcph = (const tcphdr*)(rawData +
sizeof(ether_header) + ipHeaderLen);

    uint8_t flags = ((uint8_t*)tcph)[13]; // th_flags - смещение
13 байт от начала tcphdr

    bool syn = flags & TH_SYN;
    bool fin = flags & TH_FIN;
    bool rst = flags & TH_RST;
    bool psh = flags & TH_PUSH;
    bool ack = flags & TH_ACK;
    bool urg = flags & TH_URG;

    // Несовместимые флаги: SYN+FIN, SYN+RST, FIN без ACK, URG
без ACK и т.п.

    if ((syn && fin) || (syn && rst))
    {
        if (logger)
            logger->log("Heuristic alert: Suspicious TCP flags
SYN+FIN or SYN+RST from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    if (fin && !ack)
    {
        if (logger)
            logger->log("Heuristic alert: FIN flag without ACK
from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    if (urg && !ack)
    {
        if (logger)
            logger->log("Heuristic alert: URG flag without ACK
from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
        return true;
    }

    return false;
}

bool HeuristicEngine::checkPortAnomalies(const Packet& packet)
{
    // Порты вне диапазона 1-65535 или необычные порты
(например, порты ниже 1024 при нестандартных протоколах)
    if (packet.srcPort < 0 || packet.srcPort > 65535)
    {
        if (logger)

```

```

        logger->log("Heuristic alert: Invalid source port "
+ std::to_string(packet.srcPort) + " from IP " + packet.srcIP,
"heuristic_engine", LogLevel::WARNING);
        return true;
    }
    if (packet.dstPort < 0 || packet.dstPort > 65535)
    {
        if (logger)
            logger->log("Heuristic alert: Invalid destination
port " + std::to_string(packet.dstPort) + " from IP " +
packet.srcIP, "heuristic_engine", LogLevel::WARNING);
        return true;
    }

    // Например, подозрительно много трафика на очень высокие
порты (49152-65535)
    if (packet.dstPort >= 49152 && packet.dstPort <= 65535)
    {
        if (logger)
            logger->log("Heuristic alert: Traffic to high port "
+ std::to_string(packet.dstPort) + " from IP " + packet.srcIP,
"heuristic_engine", LogLevel::WARNING);
    }

    return false;
}

bool HeuristicEngine::checkEmptyTcpPacket(const Packet& packet,
const uint8_t* rawData, size_t dataLen)
{
    // Проверяем, что TCP пакет без полезных данных (например,
длина TCP-сегмента == 0)
    if (packet.protocol != "TCP")
        return false;

    if (dataLen < sizeof(ether_header) + sizeof(ip))
        return false;

    const ip* iphdr = (const ip*)(rawData +
sizeof(ether_header));
    size_t ipHeaderLen = iphdr->ip_hl * 4;
    if (dataLen < sizeof(ether_header) + ipHeaderLen +
sizeof(tcphdr))
        return false;

    const tcphdr* tcph = (const tcphdr*)(rawData +
sizeof(ether_header) + ipHeaderLen);
    size_t tcpHeaderLen = tcph->th_off * 4;

    size_t tcpPayloadLen = ntohs(iphdr->ip_len) - ipHeaderLen -
tcpHeaderLen;

```

```

    if (tcpPayloadLen == 0)
    {
        if (logger)
            logger->log("Heuristic alert: Empty TCP packet from
IP " + packet.srcIP, "heuristic_engine", LogLevel::WARNING);
        return true;
    }
    return false;
}

bool HeuristicEngine::checkPacketSizeAnomaly(size_t dataLen)
{
    // Маленькие (< 40 байт) или очень большие (>1500 байт)
    пакеты
    if (dataLen < 40)
    {
        if (logger)
            logger->log("Heuristic alert: Very small packet
size: " + std::to_string(dataLen), "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    if (dataLen > 1500)
    {
        if (logger)
            logger->log("Heuristic alert: Very large packet
size: " + std::to_string(dataLen), "heuristic_engine",
LogLevel::WARNING);
        return true;
    }
    return false;
}

bool HeuristicEngine::checkPacketRateICMP(const Packet& packet)
{
    using clock = std::chrono::steady_clock;
    auto now = clock::now();

    auto& stats = ipStats[packet.srcIP]; // Общая статистика
    (используем ту же map)
    if (stats.packetCount == 0)
    {
        stats.firstPacketTime = now;
        stats.packetCount = 1;
        return false;
    }

    auto duration =
std::chrono::duration_cast<std::chrono::seconds>(now -
stats.firstPacketTime);
    if (duration > timeWindow)

```

```

{
    stats.packetCount = 1;
    stats.firstPacketTime = now;
    return false;
}
else
{
    stats.packetCount++;
    if (stats.packetCount > packetThreshold)
    {
        if (logger)
            logger->log("Heuristic alert: High ICMP packet
rate from IP " + packet.srcIP, "heuristic_engine",
LogLevel::WARNING);
        stats.packetCount = 0;
        return true;
    }
}
return false;
}

bool HeuristicEngine::checkIcmpAnomalies(const Packet& packet,
const uint8_t* rawData, size_t dataLen)
{
    if (packet.protocol != "ICMP")
        return false;

    if (dataLen < sizeof(ether_header) + sizeof(ip) + 4) //
Минимум для ICMP-заголовка
        return false;

    const ip* iphdr = (const ip*)(rawData +
sizeof(ether_header));
    size_t ipHeaderLen = iphdr->ip_hl * 4;

    const uint8_t* icmpData = rawData + sizeof(ether_header) +
ipHeaderLen;
    uint8_t type = icmpData[0];
    uint8_t code = icmpData[1];

    // Пример аномалий:
    // - Частые эхо-запросы
    // - Неизвестные типы (>18 по IANA)
    if (type == 8) // Echo Request (ping)
    {
        if (logger)
            logger->log("Heuristic alert: ICMP Echo Request from
IP " + packet.srcIP, "heuristic_engine", LogLevel::INFO);
    }
    else if (type > 18)
    {

```

```

        if (logger)
            logger->log("Heuristic alert: Unknown ICMP type " +
std::to_string(type) + " from IP " + packet.srcIP,
"heuristic_engine", LogLevel::WARNING);
            return true;
        }

    return false;
}

```

					ПГУ1.090301.1.001 ПЗ	Лист
						109
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.10-Листинг заголовочного файла модуля загрузки конфигурации (config_utils.h)

В данном приложении объявлены конфигурация, структура и методы для парсинга и загрузки параметров из JSON-файла.

```
#pragma once

#include <string>
#include "logger.h"

struct Config {
    std::string interface;
    LogLevel logLevel = LogLevel::INFO;
    int alertCooldown = 10;
    int heuristicPacketThreshold = 100;
    int heuristicTimeWindowSeconds = 10;
};

bool loadConfig(const std::string& filename, Config& config);
```

					ПГУ1.090301.1.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.11-Листинг файла модуля загрузки конфигурации (config_utils.cpp)

В данном приложении представлена механика для парсинга и загрузки параметров из JSON-файла.

```
#include "config_utils.h"
#include <fstream>
#include <nlohmann/json.hpp>
#include <algorithm>

using json = nlohmann::json;

static LogLevel parseLogLevel(const std::string& levelStr)
{
    std::string lvl = levelStr;
    std::transform(lvl.begin(), lvl.end(), lvl.begin(),
        ::tolower);

    if (lvl == "debug") return LogLevel::DEBUG;
    if (lvl == "info") return LogLevel::INFO;
    if (lvl == "warn" || lvl == "warning") return
LogLevel::WARNING;
    if (lvl == "error") return LogLevel::ERROR;

    return LogLevel::INFO; // default
}

bool loadConfig(const std::string& fileName, Config& config)
{
    std::ifstream file(fileName);
    if (!file.is_open())
        return false;

    try
    {
        json j;
        file >> j;

        if (j.contains("interface") &&
j["interface"].is_string())
            config.interface =
j["interface"].get<std::string>();
        else
            return false;

        if (j.contains("log_level") &&
j["log_level"].is_string())
```

```

        config.logLevel =
parseLogLevel(j["log_level"].get<std::string>());
        else
            config.logLevel = LogLevel::INFO;

        if (j.contains("alert_cooldown") &&
j["alert_cooldown"].is_number_integer())
            config.alertCooldown =
j["alert_cooldown"].get<int>();
        else
            config.alertCooldown = 10;
            if (j.contains("heuristicPacketThreshold") &&
j["heuristicPacketThreshold"].is_number_integer())
                config.heuristicPacketThreshold =
j["heuristicPacketThreshold"].get<int>();
            else
                config.heuristicPacketThreshold = 100;
            if (j.contains("heuristicTimeWindowSeconds") &&
j["heuristicTimeWindowSeconds"].is_number_integer())
                config.heuristicTimeWindowSeconds =
j["heuristicTimeWindowSeconds"].get<int>();
            else
                config.heuristicTimeWindowSeconds = 10;
        return true;
    }
    catch (...)
    {
        return false;
    }
}

```

Приложение Б.12-Листинг заголовочного файла модуля ведения логов (logger.h)

В данном приложении объявлен класс позволяющий вести логирование событий программы.

```
#pragma once

#pragma once

#include <fstream>
#include <mutex>
#include <string>
#include <chrono>
#include <unordered_map>

enum class LogLevel
{
    DEBUG = 0,
    INFO,
    WARNING,
    ERROR
};

class Logger
{
public:
    Logger(std::chrono::seconds floodInterval =
std::chrono::seconds(5), const std::string& filename =
"logs/hybridIDS.log");
    ~Logger();

    virtual void log(const std::string& message, const
std::string& module = "general", LogLevel level =
LogLevel::INFO);
    void setLogLevel(LogLevel level);
    static std::string levelToString(LogLevel level);

private:
    std::ofstream logFile;
    std::mutex logMutex;
    LogLevel currentLevel = LogLevel::INFO;

    std::string lastMessage;
    std::chrono::steady_clock::time_point lastLogTime;
    std::chrono::seconds floodInterval;
    std::unordered_map<std::string,
std::chrono::steady_clock::time_point> floodMap;
```

```
std::string getTimestamp() const;  
};
```

Приложение Б.13-Листинг файла модуля ведения логов (logger.cpp)

В данном приложении представлена реализация модуля позволяющего вести логирование событий программы.

```
#include <iostream>
#include <chrono>
#include <iomanip>
#include <ctime>
#include <regex>

#include "logger.h"

std::string extractIP(const std::string& msg) {
    static std::regex ipRegex(R"((\d{1,3}\.){3}\d{1,3})");
    std::smatch match;
    if (std::regex_search(msg, match, ipRegex)) {
        return match.str(0);
    }
    return "";
}

Logger::Logger(std::chrono::seconds floodInterval, const
std::string& filename)
    : floodInterval(floodInterval)
{
    logFile.open(filename, std::ios::app);
    if (!logFile)
    {
        std::cout << "Error opening log file: " << filename <<
std::endl;
    }
    lastLogTime = std::chrono::steady_clock::now() -
floodInterval - std::chrono::seconds(1); // чтобы первое
сообщение точно прошло
}

Logger::~~Logger()
{
    if (logFile.is_open())
    {
        logFile.close();
    }
}

void Logger::setLogLevel(LogLevel level)
{
    currentLevel = level;
}
```

```

std::string Logger::levelToString(LogLevel level)
{
    switch (level)
    {
        case LogLevel::DEBUG: return "DEBUG";
        case LogLevel::INFO: return "INFO";
        case LogLevel::WARNING: return "WARNING";
        case LogLevel::ERROR: return "ERROR";
        default: return "UNKNOWN";
    }
}

std::string Logger::getTimestamp() const
{
    auto now = std::chrono::system_clock::now();
    std::time_t nowTime =
std::chrono::system_clock::to_time_t(now);
    std::tm tm{};
    localtime_r(&nowTime, &tm);

    char buffer[20];
    std::strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S",
&tm);

    return std::string(buffer);
}

void Logger::log(const std::string& message, const std::string&
module, LogLevel level)
{
    if (level < currentLevel) return;

    std::string ip = extractIP(message);
    std::string key = levelToString(level) + "|" + ip;
    auto now = std::chrono::steady_clock::now();

    {
        std::lock_guard<std::mutex> lock(logMutex);
        if (!ip.empty() && floodMap.count(key) && now -
floodMap[key] < floodInterval)
        {
            return; // подавляем спам
        }
        floodMap[key] = now;
    }

    std::string fullMessage = "[" + getTimestamp() + "]" + " +
levelToString(level) + "]" + module + " ] " + message;

```

```
std::lock_guard<std::mutex> lock(logMutex); // новый лок для  
ВЫВОДА
```

```
if (logFile.is_open())  
{  
    logFile << fullMessage << std::endl;  
}  
  
std::cout << fullMessage << std::endl;  
}
```

					ПГУ1.090301.1.001 ПЗ	Лист
						117
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.14-Листинг заголовочного файла описания структуры сетевого пакета (packet.h)

В данном приложении описана структура сетевого пакета, используемая при анализе и фильтрации трафика.

```
#pragma once

#include <string>
#include <cstdint>

class Packet
{
public:
    std::string srcIP;
    std::string dstIP;
    uint16_t srcPort = 0;
    uint16_t dstPort = 0;
    std::string protocol;

    uint8_t icmpType = 0;
    uint8_t icmpCode = 0;

    std::string summary() const;
};
```

Приложение Б.15-Листинг файла метода формирования строки из атрибутов сетевого пакета(packet.cpp)

В данном приложении метод формирования строки из атрибутов сетевого пакета с помощью потокового вывода.

```
#include "packet.h"

#include <sstream>

std::string Packet::summary() const
{
    std::ostringstream oss;
    oss << "Protocol: " << protocol
        << ", " << srcIP << ":" << srcPort
        << " → " << dstIP << ":" << dstPort;
    return oss.str();
}
```

					ПГУ1.090301.1.001 ПЗ	Лист
						119
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.16-Листинг файла конфигурации (config.json)

В данном приложении представлена конфигурация, используемая в работе гибридной IDS.

```
{  
  "interface": "enp0s3",  
  "log_level": "info",  
  "alert_cooldown": 10,  
  "heuristicPacketThreshold": 100,  
  "heuristicTimeWindowSeconds": 10  
}
```

					ПГУ1.090301.1.001 ПЗ	Лист
						120
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.17-Листинг файла набора правил (rules.json)

В данном приложении представлен примерный набор правил, используемый для фильтрации трафика.

```
[
  {
    "id": 1,
    "description": "Блокировать все TCP пакеты с порта 80",
    "protocol": "TCP",
    "src_port": 80,
    "action": "block"
  },
  {
    "id": 2,
    "description": "Блокировать UDP трафик с IP 192.168.1.100",
    "protocol": "UDP",
    "src_ip": "192.168.1.100",
    "action": "block"
  }
]
```

					ПГУ1.090301.1.001 ПЗ	Лист
						121
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.18-Листинг файла белого списка (whitelist.txt)

В данном приложении представлен примерный набор белых адресов, которые не будут блокироваться.

```
10.0.0.1  
192.168.1.50  
127.0.0.1
```

					ПГУ1.090301.1.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.19-Листинг файла конфигурации сборки (CMakeLists.txt)

В данном приложении представлена конфигурация, содержащая ресурсы, настройки, зависимости, интеграции библиотек.

```
cmake_minimum_required(VERSION 3.20)

project(hybridIDS VERSION 1.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Let CMake find Conan packages
list(APPEND CMAKE_PREFIX_PATH "${CMAKE_BINARY_DIR}")

# Binary directory
file(MAKE_DIRECTORY ${CMAKE_SOURCE_DIR}/bin)

set(SOURCES
    src/main.cpp
    src/sniffer.cpp
    src/rule_engine.cpp
    src/nftables_control.cpp
    src/logger.cpp
    src/packet.cpp
    src/config_utils.cpp
    src/heuristic_engine.cpp
)

add_executable(hybridIDS ${SOURCES})

set_target_properties(hybridIDS PROPERTIES
    RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/bin
)

target_include_directories(hybridIDS PRIVATE
    ${CMAKE_SOURCE_DIR}/include)

# Conan-provided package
find_package(nlohmann_json REQUIRED)
target_link_libraries(hybridIDS PRIVATE
    nlohmann_json::nlohmann_json)

# System libraries
find_package(PkgConfig REQUIRED)
pkg_check_modules(PCAP REQUIRED libpcap)

target_include_directories(hybridIDS PRIVATE
    ${PCAP_INCLUDE_DIRS})
target_link_libraries(hybridIDS PRIVATE ${PCAP_LIBRARIES})
```

					ПГУ1.090301.1.001 ПЗ	Лист
						123
Изм.	Лист	№ докум.	Подпись	Дата		

```

target_compile_options(hybridIDS PRIVATE ${PCAP_CFLAGS_OTHER})

pkg_check_modules(LIBMNL REQUIRED libmnl)
pkg_check_modules(LIBNFTNL REQUIRED libnftnl)

target_include_directories(hybridIDS PRIVATE
${LIBMNL_INCLUDE_DIRS} ${LIBNFTNL_INCLUDE_DIRS})
target_link_libraries(hybridIDS PRIVATE ${LIBMNL_LIBRARIES}
${LIBNFTNL_LIBRARIES})
target_compile_options(hybridIDS PRIVATE ${LIBMNL_CFLAGS_OTHER}
${LIBNFTNL_CFLAGS_OTHER})

option(ENABLE_TESTS "Build unit tests" ON)
if(ENABLE_TESTS)
    enable_testing()
    add_subdirectory(tests)
endif()

```

					ПГУ1.090301.1.001 ПЗ	Лист
						124
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.20-Листинг конанфайла (conanfile.py)

В данном приложении представлен конан-файл, описывающий зависимости для Conan.

```
from conan import ConanFile
from conan.tools.cmake import CMake, cmake_layout

class HybridIDSConan(ConanFile):
    name = "hybridIDS"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "libpcap/1.10.1",
        "gtest/1.14.0",
        "nlohmann_json/3.11.2",
    )
    generators = "CMakeDeps", "CMakeToolchain"
    default_options = {
        "gtest/*:shared": True
    }

    def layout(self):
        cmake_layout(self)
```

					ПГУ1.090301.1.001 ПЗ	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.21-Листинг докерфайла (Dockerfile)

В данном приложении представлен докер-файл, описывающий контейнер для сборки.

```
# Using Ubuntu 22.04
FROM ubuntu:22.04

# Disable apt regime
ENV DEBIAN_FRONTEND=noninteractive

# Install libs
RUN apt-get update && apt-get install -y \
    build-essential \
    cmake \
    python3-pip \
    git \
    curl \
    iproute2 \
    iptables \
    nano \
    vim \
    pkg-config \
    libmnl-dev \
    libnftnl-dev \
    libpcap-dev \
    && pip3 install --no-cache-dir conan \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# Set the working directory
WORKDIR /app

# Copy the project into the container
COPY . .

# Build the project
RUN conan profile detect && \
    conan install . --output-folder=build --build=missing && \
    cmake -S . -B build -
DCMAKE_TOOLCHAIN_FILE=build/build/Release/generators/conan_toolc
hain.cmake -DCMAKE_BUILD_TYPE=Release && \
    cmake --build build
```

Приложение Б.22-Листинг файла автоматизации сборки и установки (build.sh)

В данном приложении представлен файл, автоматизирующий сборку, установку и очистку временных файлов.

```
#!/bin/bash
set -e

IMAGE_NAME="hybridids-build"
PROJECT_DIR="$(cd "$(dirname "$0")" && pwd)"
DEPLOY_DIR="/opt/hybridids"

echo "[build.sh]: Building a Docker image..."
docker build -t $IMAGE_NAME .

echo "[build.sh]: Starting the build inside the container..."
docker run --rm -it \
  -v "$PWD":/app \
  -w /app \
  $IMAGE_NAME \
  bash -c "
    echo '[*] Cleaning build directory...'
    rm -rf build

    echo '[*] Installing dependencies with Conan...'
    [ -f ~/.conan2/profiles/default ] || conan profile detect
    conan install . --output-folder=build --build=missing

    echo '[*] Configuring CMake with tests enabled...'
    cmake -S . -B build \
      -
    DCMAKE_TOOLCHAIN_FILE=build/build/Release/generators/conan_toolc
    hain.cmake \
      -DCMAKE_BUILD_TYPE=Release \
      -DENABLE_TESTS=ON

    echo '[*] Building the project...'
    cmake --build build

    echo '[*] Running unit tests...'
    cd build
    ctest -V --output-on-failure
  "

echo "[build.sh]: Assembly complete."

echo "[build.sh]: Installing HybridIDS in $DEPLOY_DIR..."
```

```

sudo mkdir -p "$DEPLOY_DIR/bin"
sudo mkdir -p "$DEPLOY_DIR/config"
sudo mkdir -p "$DEPLOY_DIR/logs"

sudo cp "$PROJECT_DIR/bin/hybridIDS" "$DEPLOY_DIR/bin/"

if [ -d "$PROJECT_DIR/config" ]; then
    sudo cp -r "$PROJECT_DIR/config/*" "$DEPLOY_DIR/config/"
fi

echo "[build.sh]: Create a systemd unit file..."

sudo bash -c "cat > /etc/systemd/system/hybridids.service <<EOF
[Unit]
Description=HybridIDS Intrusion Detection Service
After=network.target

[Service]
Type=simple
User=root
Group=root
WorkingDirectory=$DEPLOY_DIR
ExecStart=$DEPLOY_DIR/bin/hybridIDS
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
"

echo "[build.sh]: Updating systemd..."
sudo systemctl daemon-reload
sudo systemctl enable hybridids.service

echo "[build.sh]: Installation complete!"
echo "The service of the hybridIDS is launched as follows:"
echo "  sudo systemctl start hybridids.service"
echo "Logs can be viewed at the address:"
echo "  /opt/hybridids/logs/hibridIDS.log"

read -p "[build.sh]: Clear temporary build files (build/, bin/
and etc.)? [y/N] " confirm

if [[ "$confirm" =~ ^[Yy]$ ]]; then
    echo "[build.sh]: Deleting temporary files..."
    sudo rm -rf build/ bin/ CMakeUserPresets.json .Dockerfile.swo
.Dockerfile.swp
    echo "[build.sh]: Cleaning completed."
else

```

```
    echo "[build.sh]: Temporary files have been saved."
fi
```

Приложение Б.23-Листинг файла unit-тестов (unit_tests.cpp)

В данном приложении представлен файл, содержащий unit-тесты, для контроля за корректностью исходных файлов программы.

```
#include <gtest/gtest.h>
#include <gmock/gmock.h>
#include <netinet/ip_icmp.h>
#include <fstream>
#include <sstream>
#include <string>
#include <regex>
#include <thread>
#include <chrono>
#include <vector>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <net/ethernet.h>

#include "../include/config_utils.h"
#include "../include/heuristic_engine.h"
#include "../include/logger.h"
#include "../include/nftables_control.h"
#include "../include/rule_engine.h"
#include "../include/packet.h"
#include "../include/sniffer.h"

std::chrono::seconds floodInterval{0};

//
// ----- CONFIG_UTILS -----
//
class ConfigUtilsTest : public ::testing::Test {
protected:
    std::string validFile = "test_valid_config.json";
    std::string invalidFile = "test_invalid_config.json";

    void SetUp() override {
        std::ofstream ofs(validFile);
        ofs << R"({
            "interface": "eth0",
            "log_level": "debug",
            "alert_cooldown": 5,
            "heuristicPacketThreshold": 50,
            "heuristicTimeWindowSeconds": 5
        })";
        ofs.close();
    }
};
```

```

        std::ofstream ofs2(invalidFile);
        ofs2 << R"({
            "interface": 123,
            "log_level": 42
        })";
        ofs2.close();
    }

    void TearDown() override {
        std::remove(validFile.c_str());
        std::remove(invalidFile.c_str());
    }
};

TEST_F(ConfigUtilsTest, LoadValidConfig) {
    Config config;
    ASSERT_TRUE(loadConfig(validFile, config));
    EXPECT_EQ(config.interface, "eth0");
    EXPECT_EQ(config.logLevel, LogLevel::DEBUG);
    EXPECT_EQ(config.alertCooldown, 5);
    EXPECT_EQ(config.heuristicPacketThreshold, 50);
    EXPECT_EQ(config.heuristicTimeWindowSeconds, 5);
}

TEST_F(ConfigUtilsTest, LoadInvalidConfig) {
    Config config;
    ASSERT_FALSE(loadConfig(invalidFile, config));
}

TEST_F(ConfigUtilsTest, LoadMissingFile) {
    Config config;
    ASSERT_FALSE(loadConfig("nonexistent.json", config));
}

//
// ----- HEURISTIC_ENGINE -----
//
class DummyLogger : public Logger {
public:
    DummyLogger() : Logger(std::chrono::seconds(0), "/dev/null")
    {}

    std::vector<std::string> messages;
    void log(const std::string& msg, const std::string& src,
        LogLevel lvl) override {
        messages.push_back(msg);
    }
};

Packet makePacket(std::string proto, std::string ip =
    "192.168.1.1", int sport = 1234, int dport = 80) {
    Packet p;

```

```

    p.protocol = proto;
    p.srcIP = ip;
    p.dstIP = "10.0.0.1";
    p.srcPort = sport;
    p.dstPort = dport;
    return p;
}

TEST(HeuristicEngineTest, PacketRateThresholdTCP) {
    Config config;
    config.heuristicPacketThreshold = 3;
    config.heuristicTimeWindowSeconds = 10;

    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    std::array<uint8_t, 40> rawData = {
        // IP-заголовок (20 байт)
        0x45, 0x00, 0x00, 0x28, // Версия+длина, DSCP, длина = 40
        0x00, 0x00, 0x40, 0x00, // ID, флаги+фрагменты
        0x40, 0x06, 0x00, 0x00, // TTL, протокол (6=TCP), чек-сумма
        0x7F, 0x00, 0x00, 0x01, // Src IP: 127.0.0.1
        0x7F, 0x00, 0x00, 0x01, // Dst IP: 127.0.0.1

        // TCP-заголовок (20 байт)
        0x1F, 0x90, 0x00, 0x50, // Src port: 8080, Dst port: 80
        0x00, 0x00, 0x00, 0x00, // Seq number
        0x00, 0x00, 0x00, 0x00, // Ack number
        0x50, 0x02, 0x71, 0x10, // Header len+flags (SYN), window
        0x00, 0x00, 0x00, 0x00 // Checksum, urgent pointer
    };

    Packet pkt = makePacket("TCP");

    for (int i = 0; i < 4; ++i)
    {
        engine.analyzePacket(pkt, rawData.data(), 100);
    }

    std::this_thread::sleep_for(std::chrono::milliseconds(2));

    EXPECT_FALSE(logger.messages.empty());
}

TEST(HeuristicEngineTest, EmptyUdpPacketTriggersAlert) {
    Config config;
    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    uint8_t raw[64] = {};
    Packet p = makePacket("UDP");

```

```

        ip* iphdr = (ip*)(raw + sizeof(ether_header));
        iphdr->ip_hl = 5;
        iphdr->ip_len = htons(28);
        udphdr* udph = (udphdr*)(raw + sizeof(ether_header) +
sizeof(ip));
        udph->len = htons(8);

        engine.analyzePacket(p, raw, sizeof(raw));
        ASSERT_FALSE(logger.messages.empty());
    }

TEST(HeuristicEngineTest, SuspiciousTcpFlagsDetected) {
    Config config;
    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    uint8_t raw[64] = {};
    Packet p = makePacket("TCP");

    ip* iphdr = (ip*)(raw + sizeof(ether_header));
    iphdr->ip_hl = 5;
    iphdr->ip_len = htons(40);
    tcphdr* tcph = (tcphdr*)(raw + sizeof(ether_header) +
sizeof(ip));
    tcph->th_off = 5;
    ((uint8_t*)tcph)[13] = TH_SYN | TH_FIN;

    std::this_thread::sleep_for(std::chrono::milliseconds(1100));
    engine.analyzePacket(p, raw, sizeof(raw));
    ASSERT_FALSE(logger.messages.empty());
}

TEST(HeuristicEngineTest, EmptyTcpPayloadDetected) {
    Config config;
    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    uint8_t raw[64] = {};
    Packet p = makePacket("TCP");

    ip* iphdr = (ip*)(raw + sizeof(ether_header));
    iphdr->ip_hl = 5;
    iphdr->ip_len = htons(40);
    tcphdr* tcph = (tcphdr*)(raw + sizeof(ether_header) +
sizeof(ip));
    tcph->th_off = 5;

    std::this_thread::sleep_for(std::chrono::milliseconds(5500));
    engine.analyzePacket(p, raw, sizeof(raw));

```

```

        ASSERT_FALSE(logger.messages.empty());
    }

TEST(HeuristicEngineTest, IcmpPacketTriggersAlertOnEmptyPayload)
{
    Config config;
    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    uint8_t raw[64] = {};
    Packet p = makePacket("ICMP");

    // IP-заголовок
    ip* iphdr = (ip*)(raw + sizeof(ether_header));
    iphdr->ip_hl = 5;
    iphdr->ip_len = htons(28);
    iphdr->ip_p = IPPROTO_ICMP;

    // ICMP-заголовок
    icmphdr* icmph = (icmphdr*)(raw + sizeof(ether_header) +
sizeof(ip));
    icmph->type = 8; // Echo request
    icmph->code = 0;
    icmph->checksum = 0; // Можно посчитать, но обычно для
теста можно 0

    engine.analyzePacket(p, raw, 28); // Передаем длину IP+ICMP
без лишнего мусора

    ASSERT_FALSE(logger.messages.empty());
}

TEST(HeuristicEngineTest,
IcmpPacketWithSuspiciousTypeTriggersAlert) {
    Config config;
    DummyLogger logger;
    HeuristicEngine engine(config, &logger);

    uint8_t raw[64] = {};
    Packet p = makePacket("ICMP");

    ip* iphdr = (ip*)(raw + sizeof(ether_header));
    iphdr->ip_hl = 5;
    iphdr->ip_len = htons(28);
    iphdr->ip_p = IPPROTO_ICMP;

    icmphdr* icmph = (icmphdr*)(raw + sizeof(ether_header) +
sizeof(ip));
    icmph->type = 3; // Destination unreachable

```

```

        icmph->code = 13; // Communication administratively
        prohibited
        icmph->checksum = 0;

std::this_thread::sleep_for(std::chrono::milliseconds(5500));
engine.analyzePacket(p, raw, 28);

    ASSERT_FALSE(logger.messages.empty());
}

//
// ----- LOGGER -----
//
class LoggerTest : public ::testing::Test {
protected:
    const std::string testLogFile = "test_logger_output.log";

    void TearDown() override {
        std::remove(testLogFile.c_str());
    }

    std::string readLogFile() {
        std::ifstream file(testLogFile);
        std::stringstream buffer;
        buffer << file.rdbuf();
        return buffer.str();
    }
};

TEST_F(LoggerTest, LogWritesToFileAndConsole) {
    Logger logger(floodInterval, testLogFile);
    logger.setLogLevel(LogLevel::DEBUG);

    std::string message = "Test message";
    logger.log(message, "test_module", LogLevel::INFO);

    std::string content = readLogFile();
    EXPECT_NE(content.find("Test message"), std::string::npos);
    EXPECT_NE(content.find("test_module"), std::string::npos);
    EXPECT_NE(content.find("INFO"), std::string::npos);
}

TEST_F(LoggerTest, LogLevelFiltering) {
    Logger logger(floodInterval, testLogFile);
    logger.setLogLevel(LogLevel::WARNING);

    logger.log("This should not appear", "test",
    LogLevel::INFO);
    logger.log("This should appear", "test", LogLevel::ERROR);

    std::string content = readLogFile();

```

```

    EXPECT_EQ(content.find("This should not appear"),
std::string::npos);
    EXPECT_NE(content.find("This should appear"),
std::string::npos);
}

TEST_F(LoggerTest, TimestampFormat) {
    Logger logger(floodInterval, testLogFile);
    logger.setLogLevel(LogLevel::DEBUG);
    logger.log("Timestamp test", "time", LogLevel::INFO);

    std::string content = readLogFile();
    std::regex pattern(R"(\d{4}-\d{2}-\d{2}
\d{2}:\d{2}:\d{2})\[INFO\]\[time\] Timestamp test)");
    EXPECT_TRUE(std::regex_search(content, pattern));
}

TEST_F(LoggerTest, LevelToStringReturnsCorrectString) {
    EXPECT_EQ(Logger::levelToString(LogLevel::DEBUG), "DEBUG");
    EXPECT_EQ(Logger::levelToString(LogLevel::INFO), "INFO");
    EXPECT_EQ(Logger::levelToString(LogLevel::WARNING),
"WARNING");
    EXPECT_EQ(Logger::levelToString(LogLevel::ERROR), "ERROR");
}

//
// ----- NFTABLES_CONTROL -----
//
class NftablesControlTest : public ::testing::Test {
protected:
    DummyLogger dummyLogger;
};

TEST_F(NftablesControlTest, Initialization) {
    NftablesControl nft(&dummyLogger);
    EXPECT_TRUE(nft.init());
}

TEST_F(NftablesControlTest, BlockValidIP) {
    NftablesControl nft(&dummyLogger);
    ASSERT_TRUE(nft.init());
    EXPECT_TRUE(nft.blockIP("192.168.1.1"));
}

TEST_F(NftablesControlTest, BlockInvalidIP) {
    NftablesControl nft(&dummyLogger);
    ASSERT_TRUE(nft.init());
    EXPECT_TRUE(nft.blockIP("256.256.256.256"));
}

//

```

```

// ----- RULE_ENGINE -----
//
class RuleEngineTest : public ::testing::Test {
protected:
    DummyLogger dummyLogger;
    Config config{.alertCooldown = 0};
    std::string rulesFile = "test_rules.json";
    std::string whitelistFile = "test_whitelist.txt";

    void SetUp() override {
        std::ofstream ofs(rulesFile);
        ofs << R"([
            {
                "src_ip": "1.2.3.4",
                "dst_port": 80,
                "protocol": "TCP"
            }
        ])" ;
        ofs.close();

        std::ofstream wl(whitelistFile);
        wl << "5.6.7.8\n";
        wl.close();
    }

    void TearDown() override {
        std::remove(rulesFile.c_str());
        std::remove(whitelistFile.c_str());
    }
};

TEST_F(RuleEngineTest, LoadValidRules) {
    RuleEngine engine(config, &dummyLogger);
    EXPECT_TRUE(engine.loadRules(rulesFile));
}

TEST_F(RuleEngineTest, LoadInvalidRulesFile) {
    RuleEngine engine(config, &dummyLogger);
    EXPECT_FALSE(engine.loadRules("nonexistent.json"));
}

TEST_F(RuleEngineTest, LoadWhitelist) {
    RuleEngine engine(config, &dummyLogger);
    EXPECT_TRUE(engine.loadWhitelist(whitelistFile));
}

TEST_F(RuleEngineTest, CheckPacketMatchesRule) {
    RuleEngine engine(config, &dummyLogger);
    engine.loadRules(rulesFile);

    Packet p = makePacket("TCP", "1.2.3.4", 12345, 80);

```

```

        EXPECT_TRUE(engine.checkPacket(p));
    }

    TEST_F(RuleEngineTest, CheckPacketDoesNotMatchRule) {
        RuleEngine engine(config, &dummyLogger);
        engine.loadRules(rulesFile);

        Packet p = makePacket("TCP", "9.9.9.9", 12345, 443);
        EXPECT_FALSE(engine.checkPacket(p));
    }

    TEST_F(RuleEngineTest, WhitelistSkipsDetection) {
        RuleEngine engine(config, &dummyLogger);
        engine.loadRules(rulesFile);
        engine.loadWhitelist(whitelistFile);

        Packet p = makePacket("TCP", "5.6.7.8", 12345, 80);
        EXPECT_FALSE(engine.checkPacket(p));
    }

    TEST_F(RuleEngineTest, CheckIcmpPacketMatchesRule) {
        // Добавим правило для ICMP
        std::ofstream ofs(rulesFile);
        ofs << R"([
            {
                "src_ip": "1.2.3.4",
                "protocol": "ICMP"
            }
        ])" ;
        ofs.close();

        RuleEngine engine(config, &dummyLogger);
        EXPECT_TRUE(engine.loadRules(rulesFile));

        Packet icmpPkt = makePacket("ICMP", "1.2.3.4");
        EXPECT_TRUE(engine.checkPacket(icmpPkt));
    }

    TEST_F(RuleEngineTest, CheckIcmpPacketDoesNotMatchRule) {
        std::ofstream ofs(rulesFile);
        ofs << R"([
            {
                "src_ip": "1.2.3.4",
                "protocol": "ICMP"
            }
        ])" ;
        ofs.close();

        RuleEngine engine(config, &dummyLogger);
        EXPECT_TRUE(engine.loadRules(rulesFile));
    }

```

```

        Packet icmpPkt = makePacket("ICMP", "9.9.9.9");
        EXPECT_FALSE(engine.checkPacket(icmpPkt));
    }

    TEST_F(RuleEngineTest, IcmpPacketWhitelisted) {
        std::ofstream ofs(rulesFile);
        ofs << R"([
            {
                "src_ip": "5.6.7.8",
                "protocol": "ICMP"
            }
        ])" ;
        ofs.close();

        RuleEngine engine(config, &dummyLogger);
        EXPECT_TRUE(engine.loadRules(rulesFile));
        engine.loadWhitelist(whitelistFile); // whitelist содержит
        5.6.7.8

        Packet icmpPkt = makePacket("ICMP", "5.6.7.8");
        EXPECT_FALSE(engine.checkPacket(icmpPkt));
    }

    //
    // ----- SNIFFER -----
    //
    class SnifferTest : public ::testing::Test {
    protected:
        DummyLogger logger;
    };

    TEST_F(SnifferTest, ConstructAndSetHandler) {
        Sniffer sniffer("lo", &logger);
        bool called = false;
        sniffer.setPacketHandler([&](const Packet& p, const
        uint8_t*, size_t) {
            called = true;
        });
        EXPECT_FALSE(called);
    }

    TEST_F(SnifferTest, StartStopSnifferNoCrash) {
        Sniffer sniffer("lo", &logger);
        sniffer.start();
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
        sniffer.stop();
        SUCCEED();
    }

    TEST_F(SnifferTest, InvalidInterfaceLogsError) {

```

```

Sniffer sniffer("invalid0", &logger);
sniffer.start();
std::this_thread::sleep_for(std::chrono::milliseconds(100));
sniffer.stop();

bool errorLogged = false;
for (const auto& line : logger.messages) {
    if (line.find("Error opening interface") !=
std::string::npos) {
        errorLogged = true;
        break;
    }
}
EXPECT_TRUE(errorLogged);
}

TEST_F(SnifferTest, ParseIcmpPacket) {
    // Формируем минимальный Ethernet + IP + ICMP пакет (echo
request)

    // Ethernet (14 байт)
    std::array<uint8_t, 14> eth = {
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff, // MAC dst (broadcast)
        0x00, 0x0c, 0x29, 0xab, 0xcd, 0xef, // MAC src (пример)
        0x08, 0x00 // Ethertype = IPv4
(0x0800)
    };

    // IP (20 байт)
    std::array<uint8_t, 20> ip = {
        0x45, 0x00, 0x00, 0x1c, // Версия+hlen=5(20байт), DSCP,
длина=28 байт
        0x00, 0x00, 0x00, 0x00, // ID, Flags, Fragment offset
        64, IPPROTO_ICMP, 0x00, 0x00, // TTL=64,
протокол=1(ICMP), контрольная сумма (0 для теста)
        127, 0, 0, 1, // Src IP = 127.0.0.1
        127, 0, 0, 1 // Dst IP = 127.0.0.1
    };

    // ICMP (8 байт минимум, но нам хватит 2 для типа и кода)
    std::array<uint8_t, 8> icmp = {
        8, 0, // Тип = 8 (echo request), Код = 0
        0x00, 0x00, // Чек-сумма (0 для теста)
        0x00, 0x01, 0x00, 0x01 // Остальные поля (идентификатор,
seq)
    };

    // Собираем полный пакет
    std::vector<uint8_t> packet;
    packet.insert(packet.end(), eth.begin(), eth.end());
    packet.insert(packet.end(), ip.begin(), ip.end());

```

```

packet.insert(packet.end(), icmp.begin(), icmp.end());

// Создаём Sniffer с тестовым логгером
DummyLogger logger;
Sniffer sniffer("lo", &logger);

Packet parsedPacket;
bool handlerCalled = false;

sniffer.setPacketHandler([&](const Packet& p, const uint8_t*
data, size_t len) {
    parsedPacket = p;
    handlerCalled = true;
});

// Имитация вызова обработки пакета вручную (т.к. pcap не
используется в тестах)

// Разбор Ethernet
if (packet.size() < sizeof(ether_header)) FAIL();
const struct ether_header* ethh = (const struct
ether_header*)packet.data();
if (ntohs(ethh->ether_type) != ETHERTYPE_IP) FAIL();

if (packet.size() < sizeof(ether_header) + sizeof(ip))
FAIL();
const struct ip* iphdr = (const struct ip*)(packet.data() +
sizeof(ether_header));
Packet pkt;
pkt.srcIP = inet_ntoa(iphdr->ip_src);
pkt.dstIP = inet_ntoa(iphdr->ip_dst);

const uint8_t* transportHeader = packet.data() +
sizeof(ether_header) + iphdr->ip_hl * 4;

switch (iphdr->ip_p)
{
    case IPPROTO_ICMP:
        pkt.protocol = "ICMP";
        pkt.icmpType = transportHeader[0];
        pkt.icmpCode = transportHeader[1];
        break;
    default:
        FAIL();
}

parsedPacket = pkt;
handlerCalled = true;

ASSERT_TRUE(handlerCalled);
EXPECT_EQ(parsedPacket.protocol, "ICMP");

```

```

    EXPECT_EQ(parsedPacket.icmpType, 8);
    EXPECT_EQ(parsedPacket.icmpCode, 0);
    EXPECT_EQ(parsedPacket.srcIP, "127.0.0.1");
    EXPECT_EQ(parsedPacket.dstIP, "127.0.0.1");
}

TEST_F(SnifferTest, ParseTcpPacket) {
    // Ethernet
    std::array<uint8_t, 14> eth = {
        0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
        0x00, 0x0c, 0x29, 0xab, 0xcd, 0xef,
        0x08, 0x00
    };
    // IP
    std::array<uint8_t, 20> ip = {
        0x45, 0x00, 0x00, 0x28, // длина 40 байт (20 IP + 20
TCP)
        0x00, 0x00, 0x00, 0x00,
        64, IPPROTO_TCP, 0x00, 0x00,
        192, 168, 0, 1,
        192, 168, 0, 2
    };
    // TCP (20 байт)
    std::array<uint8_t, 20> tcp = {
        0x04, 0xd2, 0x00, 0x50, // src port = 1234, dst port =
80
        0x00, 0x00, 0x00, 0x00, // seq number
        0x00, 0x00, 0x00, 0x00, // ack number
        0x50, 0x02, // data offset=5 (20 байт),
flags SYN
        0x72, 0x10, // window
        0x00, 0x00, // checksum
        0x00, 0x00 // urgent pointer
    };
    std::vector<uint8_t> packet;
    packet.insert(packet.end(), eth.begin(), eth.end());
    packet.insert(packet.end(), ip.begin(), ip.end());
    packet.insert(packet.end(), tcp.begin(), tcp.end());

    DummyLogger logger;
    Sniffer sniffer("lo", &logger);

    Packet parsedPacket;
    bool handlerCalled = false;

    sniffer.setPacketHandler([&](const Packet& p, const
uint8_t*, size_t) {
        parsedPacket = p;
        handlerCalled = true;
    });
}

```

```

        if (packet.size() < sizeof(ether_header)) FAIL();
        const struct ether_header* ethh = (const struct
ether_header*)packet.data();
        if (ntohs(ethh->ether_type) != ETHERTYPE_IP) FAIL();
        if (packet.size() < sizeof(ether_header) + sizeof(ip))
FAIL();

        const struct ip* iphdr = (const struct ip*)(packet.data() +
sizeof(ether_header));
        Packet pkt;
        pkt.srcIP = inet_ntoa(iphdr->ip_src);
        pkt.dstIP = inet_ntoa(iphdr->ip_dst);

        const uint8_t* transportHeader = packet.data() +
sizeof(ether_header) + iphdr->ip_hl * 4;

        switch (iphdr->ip_p)
        {
            case IPPROTO_TCP:
            {
                pkt.protocol = "TCP";
                const struct tcphdr* tcph = (const struct
tcphdr*)transportHeader;
                pkt.srcPort = ntohs(tcph->th_sport);
                pkt.dstPort = ntohs(tcph->th_dport);
                break;
            }
            default:
                FAIL();
        }
        parsedPacket = pkt;
        handlerCalled = true;

        ASSERT_TRUE(handlerCalled);
        EXPECT_EQ(parsedPacket.protocol, "TCP");
        EXPECT_EQ(parsedPacket.srcPort, 1234);
        EXPECT_EQ(parsedPacket.dstPort, 80);
        EXPECT_EQ(parsedPacket.srcIP, "192.168.0.1");
        EXPECT_EQ(parsedPacket.dstIP, "192.168.0.2");
    }

    TEST_F(SnifferTest, ParseUdpPacket) {
        // Ethernet
        std::array<uint8_t, 14> eth = {
            0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
            0x00, 0x0c, 0x29, 0xab, 0xcd, 0xef,
            0x08, 0x00
        };
        // IP
        std::array<uint8_t, 20> ip = {

```

```

        0x45, 0x00, 0x00, 0x1c, // длина 28 байт (20 IP + 8
UDP)
        0x00, 0x00, 0x00, 0x00,
        64, IPPROTO_UDP, 0x00, 0x00,
        10, 0, 0, 1,
        10, 0, 0, 2
    };
    // UDP (8 байт)
    std::array<uint8_t, 8> udp = {
        0x1f, 0x90, 0x00, 0x35, // src port=8080, dst port=53
        0x00, 0x08, 0x00, 0x00 // length=8, checksum=0
    };

    std::vector<uint8_t> packet;
    packet.insert(packet.end(), eth.begin(), eth.end());
    packet.insert(packet.end(), ip.begin(), ip.end());
    packet.insert(packet.end(), udp.begin(), udp.end());

    DummyLogger logger;
    Sniffer sniffer("lo", &logger);

    Packet parsedPacket;
    bool handlerCalled = false;

    sniffer.setPacketHandler([&](const Packet& p, const
uint8_t*, size_t) {
        parsedPacket = p;
        handlerCalled = true;
    });

    if (packet.size() < sizeof(ether_header)) FAIL();
    const struct ether_header* ethh = (const struct
ether_header*)packet.data();
    if (ntohs(ethh->ether_type) != ETHERTYPE_IP) FAIL();
    if (packet.size() < sizeof(ether_header) + sizeof(ip))
FAIL();

    const struct ip* iphdr = (const struct ip*)(packet.data() +
sizeof(ether_header));
    Packet pkt;
    pkt.srcIP = inet_ntoa(iphdr->ip_src);
    pkt.dstIP = inet_ntoa(iphdr->ip_dst);

    const uint8_t* transportHeader = packet.data() +
sizeof(ether_header) + iphdr->ip_hl * 4;

    switch (iphdr->ip_p)
    {
        case IPPROTO_UDP:
        {
            pkt.protocol = "UDP";

```

```

        const struct udphdr* udph = (const struct
udphdr*)transportHeader;
        pkt.srcPort = ntohs(udph->uh_sport);
        pkt.dstPort = ntohs(udph->uh_dport);
        break;
    }
    default:
        FAIL();
}
parsedPacket = pkt;
handlerCalled = true;

ASSERT_TRUE(handlerCalled);
EXPECT_EQ(parsedPacket.protocol, "UDP");
EXPECT_EQ(parsedPacket.srcPort, 8080);
EXPECT_EQ(parsedPacket.dstPort, 53);
EXPECT_EQ(parsedPacket.srcIP, "10.0.0.1");
EXPECT_EQ(parsedPacket.dstIP, "10.0.0.2");
}

int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}

```

Приложение Б.24-Листинг CMake-файла unit-тестов (CMakeLists.cpp)

В данном приложении представлен файл, содержащий конфигурацию unit-тестов.

```
find_package(GTest REQUIRED)
find_package(nlohmann_json REQUIRED)
find_package(PkgConfig REQUIRED)

pkg_check_modules(PCAP REQUIRED libpcap)
pkg_check_modules(LIBMNL REQUIRED libmnl)
pkg_check_modules(LIBNFTNL REQUIRED libnftnl)

add_executable(unit_tests
    ../src/logger.cpp
    ../src/sniffer.cpp
    ../src/nftables_control.cpp
    ../src/rule_engine.cpp
    ../src/heuristic_engine.cpp
    ../src/config_utils.cpp
    ../src/packet.cpp
    unit_tests.cpp
)

target_compile_definitions(unit_tests PRIVATE TESTING)

target_include_directories(unit_tests PRIVATE
    ${PROJECT_SOURCE_DIR}/include
    ${PCAP_INCLUDE_DIRS}
    ${LIBMNL_INCLUDE_DIRS}
    ${LIBNFTNL_INCLUDE_DIRS}
)

target_link_libraries(unit_tests
    GTest::gtest
    GTest::gtest_main
    nlohmann_json::nlohmann_json
    pthread

    ${PCAP_LIBRARIES}
    ${LIBMNL_LIBRARIES}
    ${LIBNFTNL_LIBRARIES}
)

add_test(NAME AllUnitTests COMMAND unit_tests --gtest_color=yes
--gtest_print_time=1)
```

Приложение В

(Обязательно)

Гибридная система анализа трафика и управления nftables

Презентация

Гибридная система анализа сетевого трафика и управления nftables

Автор работы: Аверочкин Павел Сергеевич

Научный уководитель: доцент кафедры ВТ Сорокин А.А.

Пенза, 2025 год

Актуальность темы

- Рост сетевых угроз и кибератак
- Недостатки сигнатурных и эвристических IDS/IPS
- Потребность в легких и эффективных решениях

					ПГУ1.090301.1.001 ПЗ	Лист
						147
Изм.	Лист	№ докум.	Подпись	Дата		

Цель и задачи

Цель работы: Разработка гибридной IDS/IPS системы для Linux.

Задачи:

- Анализ аналогов;
- Разработка архитектуры;
- Реализация компонентов;
- Интеграция с nftables;
- Тестирование и отладка.

Анализ аналогов

Система	Плюсы	Минусы	Тип
Snort	Большая база, поддержка многих протоколов	Нет эвристики, ресурсоемкая	Зарубежная (open-source)
Suricata	Быстрая, многопоточная	Сложная настройка	Зарубежная (open-source)
Zeek	Гибкость, поведенческий анализ	Нет сигнатур по умолчанию	Зарубежная (open-source)
СТАФОР	Сертификация ФСТЭК, GUI	Закрытый код	Российская (коммерческая)
Континент IPS	DPI, сигнатуры	Высокая стоимость	Российская (коммерческая)
Attack Killer	Машинное обучение, облачная архитектура	Зависимость от облака, для крупного бизнеса	Российская (облачная)

Архитектура HybridIDS

Компоненты:

- Sniffer → Rule & Heuristic Engine
- Logger
- Nftables Control
- Config Utils
- Packet

Ключевые моменты:

- JSON-конфигурация
- Белый список IP
- Работа в реальном времени

Технологический стек



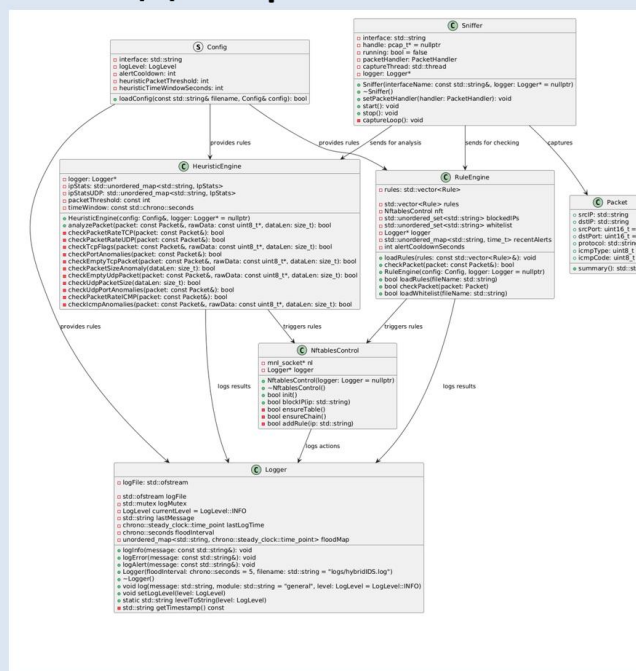
- Языки: C++, Python, Shell
- Сборка: CMake + Conan
- Контейнеризация: Docker
- ОС: Ubuntu Server
- Среда: VSCode + SSH
- Библиотеки: libpcap, libmnl



Реализация системы

- Сигнатурный анализ: rules.json
- Эвристический анализ: TCP/UDP/ICMP поведение
- Логирование событий
- Блокировка через nftables

UML-диаграмма классов



Пример конфигурации

- {
- "interface": "enp0s3",
- "log_level": "info",
- "alert_cooldown": 10,
- "heuristicPacketThreshold": 100,
- "heuristicTimeWindowSeconds": 10
- }

Система «HybridIDS»

```
admin@HybridIDS:~/hybridIDS$ sudo systemctl status hybridids.service
hybridids.service - HybridIDS Intrusion Detection Service
Loaded: loaded (/etc/systemd/system/hybridids.service; enabled; preset: enabled)
Active: inactive (dead) since Sun 2025-06-08 01:58:45 UTC; 12min ago
Duration: 1min 40.127s
Main PID: 631 (code=exited, status=0/SUCCESS)
CPU: 23ms

Jun 08 01:57:42 HybridIDS hybridIDS(631): [2025-06-08 01:57:42][WARNING][heuristic_engine] Heuristic alert: Very large UDP packet size: 642
Jun 08 01:57:42 HybridIDS hybridIDS(631): [2025-06-08 01:57:42][WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 151.0.2.53
Jun 08 01:58:13 HybridIDS hybridIDS(631): [2025-06-08 01:58:13][WARNING][heuristic_engine] Heuristic alert: UDP traffic on system port from IP 132.169.31.176
Jun 08 01:58:14 HybridIDS hybridIDS(631): [2025-06-08 01:58:14][WARNING][heuristic_engine] Heuristic alert: Traffic to high port 54173 from IP 151.0.2.53
Jun 08 01:58:45 HybridIDS hybridIDS(631): [2025-06-08 01:58:45][INFO][main] Termination signal received (15)
Jun 08 01:58:45 HybridIDS systemd[1]: Stopping hybridids.service - HybridIDS Intrusion Detection Service...
Jun 08 01:58:45 HybridIDS hybridIDS(631): [2025-06-08 01:58:45][INFO][main] Stop hybridIDS
Jun 08 01:58:45 HybridIDS hybridIDS(631): [2025-06-08 01:58:45][INFO][main] hybridIDS completed
Jun 08 01:58:45 HybridIDS systemd[1]: hybridids.service: Deactivated successfully.
Jun 08 01:58:45 HybridIDS systemd[1]: Stopped hybridids.service - HybridIDS Intrusion Detection Service.
admin@HybridIDS:~/hybridIDS$
```

```
Система «HybridIDS»
[2025-06-08 02:12:52][INFO][main] Logging level: INFO
[2025-06-08 02:12:52][INFO][main] Interface: enp0s3
[2025-06-08 02:12:52][INFO][main] Starting hybridIDS
[2025-06-08 02:12:52][INFO][rule_engine] Rules loaded successfully: 2
[2025-06-08 02:12:52][INFO][rule_engine] Loaded whitelist from file: config/whitelist.txt
```

```
Система «HybridIDS»
{
  "interface": "enp0s3",
  "log_level": "info",
  "alert_cooldown": 10,
  "heuristicPacketThreshold": 100,
  "heuristicTimeWindowSeconds": 10
}
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.1.001 ПЗ

Лист

151

Unit Тестирование

Покрытые модули:

- Конфигурация
- Эвристика
- Сигнатуры
- Сниффер
- nftables
- Логирование

Инструменты: Google Test, GMock

[illegible][illegible]

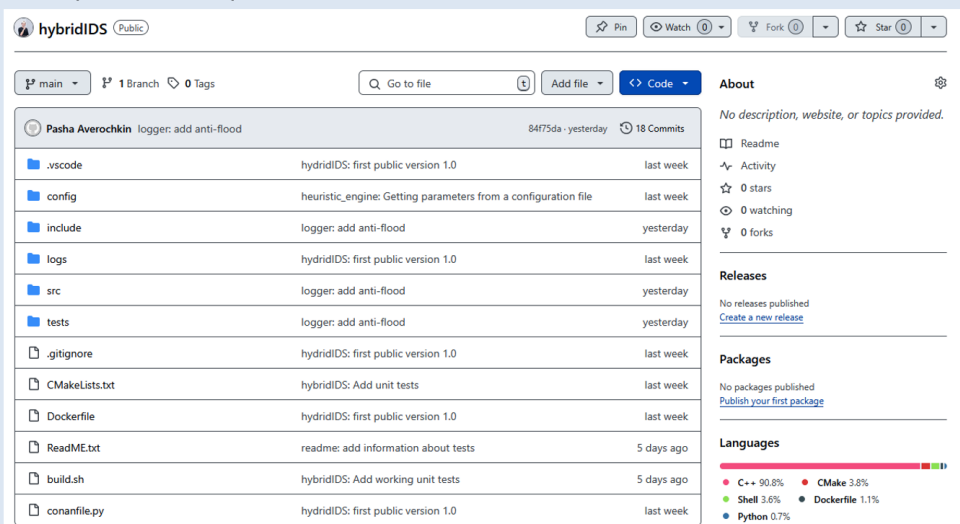
Практическое тестирование

1. **Пустой UDP-запрос на DNS-порт 53**
Эмуляция запроса без полезной нагрузки, часто используемая в DDoS-атаках **amplification**.
2. **SYN-флуд (SYN flood)**
Множество незавершённых TCP-соединений, перегружающих сервер.
3. **Аномальные комбинации TCP-флагов (SYN + FIN одновременно)**
Используются для обхода фильтров и создания нестандартного трафика.
4. **АСК-пакеты без установленного соединения**
Применяются для разведки и сканирования портов.
5. **UDP-флуд с высокой частотой (500 пакетов в секунду)**
Проверка способности системы блокировать UDP-флуд.
6. **Повторная отправка небольших UDP-запросов на порт 53**
Типичный метод **DDoS**, проверка фильтрации по порту и объёму.
7. **ICMP-флуд (Echo-запросы с высокой частотой)**
Проверка обнаружения и блокировки чрезмерной активности.
8. **ICMP-пакет с минимальной полезной нагрузкой**
Проверка реакции на нестандартно малые пакеты.
9. **Oversized ICMP-пакеты**
Тест обработки больших пакетов, способных вызвать фрагментацию или переполнение.
10. **Stealth-сканирование TCP (TCP SYN scan)**
Скрытое сканирование портов с отправкой только SYN-пакетов.
11. **Легитимные одиночные ICMP-запросы**
Проверка, что они не блокируются ошибочно.
12. **ICMP Echo-запросы с интервалом 1 мс (ICMP flood)**
Моделирование высокочастотного ICMP-флуда.
13. **Блокировка TCP-трафика с определённого IP-адреса**
Тестирование работы белого и чёрного списков.
14. **Атака с IP-адреса из белого списка**
Проверка, что сигнатурный движок не блокирует трафик с доверенных адресов.

Изм.	Лист	№ докум.	Подпись	Дата

Результаты

- Разработана модульная гибридная IDS/IPS
- Поддержка блокировки в реальном времени
- Высокая производительность
- Простота настройки



Выводы

- Выявление известных и новых угроз
- Легкая интеграция и масштабирование
- Надежная работа в условиях ограниченных ресурсов

- Текст слайда

Благодарю за внимание



					ПГУ1.090301.1.001 ПЗ	Лист
						154
Изм.	Лист	№ докум.	Подпись	Дата		