**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный университет**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**отчет**

**по лабораторной работе №2**

**по дисциплине «Системное программирование в Linux»**

**на тему «Обнаружение и блокировка подозрительного сетевого трафика»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 24Б15-пу |  | Шаймиев К. А. |
| Преподаватель |  | Киямов Ж. У. |

**Санкт-Петербург**

**2025 г**

Оглавление

[Цель работы 3](#_Toc211905320)

[Формализация задачи 3](#_Toc211905321)

[Основные шаги программы 5](#_Toc211905322)

[Блок-схема программы 6](#_Toc211905323)

[Описание программы 7](#_Toc211905324)

[Контрольный пример 11](#_Toc211905325)

[Рекомендации пользователя 14](#_Toc211905326)

[Рекомендации для программиста 14](#_Toc211905327)

[Исслндование 15](#_Toc211905328)

[Вывод 16](#_Toc211905329)

[Список использованной литературы 16](#_Toc211905330)

# **Цель работы**

Целью данной лабораторной работы является разработка программы, которая в реальном времени анализирует сетевые пакеты, регистрирует подозрительные события по набору правил, блокирует нарушителей (как точечно, так и глобально при всплеске источников) и ведёт журнал логов.

# **Формализация задачи**

Задача заключается в создании программы для обнаружения и блокировки сетевого трафика с реализацией следующих функциональных требований:

1. **Слушать трафик с помощью Scapy**: В программе должен быть компонент, который будет слушать и анализировать трафик с помощью библиотеки Scapy.
2. **Создать набор правил**: Необходимо задать правила, по которым можно будет определить, какой трафик считать подозрительным.
3. **Фиксировать обнаружение в журнал**: Программа должна записывать в журнал логи – обнаружение подозрительного трафика, DDoS атак и т.д.
4. **Возможность блокировки**: В программе должна быть возможность блокировки трафика от теоретически опасных адресов и механизм противостояния DDoS атакам.
5. **Тестирование и настройка**: Программа должна быть пригодной к тестированию и позволять настраивать параметры своей работы.
6. **Графический интерфейс**: Необходимо реализовать графический интерфейс для взаимодействия с программой, который будет включать текущий статус детектора, список адресов от которых приходят пакеты, возможность блокировки выбранных адресов и параметры работы детектора.

**Теоретическая часть**

Обнаружение сетевых аномалий — ключевая задача ИБ. В условиях постоянного роста трафика требуется механизмы автоматизации и потоковой обработки пакетов.

Основные концепции, применяемые при разработке программы для анализа:

1. **Сниффинг и предварительная обработка**

Сбор пакетов ведётся на заданном интерфейсе с фильтрами BPF (например, ip). Фильтр отбрасывает «лишнее» ещё в ядре, снижая нагрузку. Обработка идёт без хранения пакетов (streaming, store=False).:

* Интерфейс и BPF задаются из GUI/CLI;
* В Пакеты передаются в конвейер детекции в реальном времени.

В работе применяется библиотека Scapy, предоставляющая широкий набор инструментов для решения задач, связанных с сетями.

1. **Извлечение признаков и агрегирование.**

Из каждого пакета берутся поля src/dst, proto, dport (TCP/UDP), length, метка времени. Ведётся состояние по источникам:

* История прибытия пакетов для каждого src
* Последнее появление src для подсчета уникальных адресов в окне
* Счётчики по IP для простых частотных правил.

1. **Правила определения и признаки подозрительного трафика**

Комбинируются сигнатурные и поведенческие подходы:

* Высокая частота пакетов от одного источника (триггер на каждый N-й пакет);
* Нетипичный порт назначения (не в наборе распространённых);
* Всплеск числа уникальных источников за короткое окно (индикатор DDOS).

1. **Реакции и управление доступом**

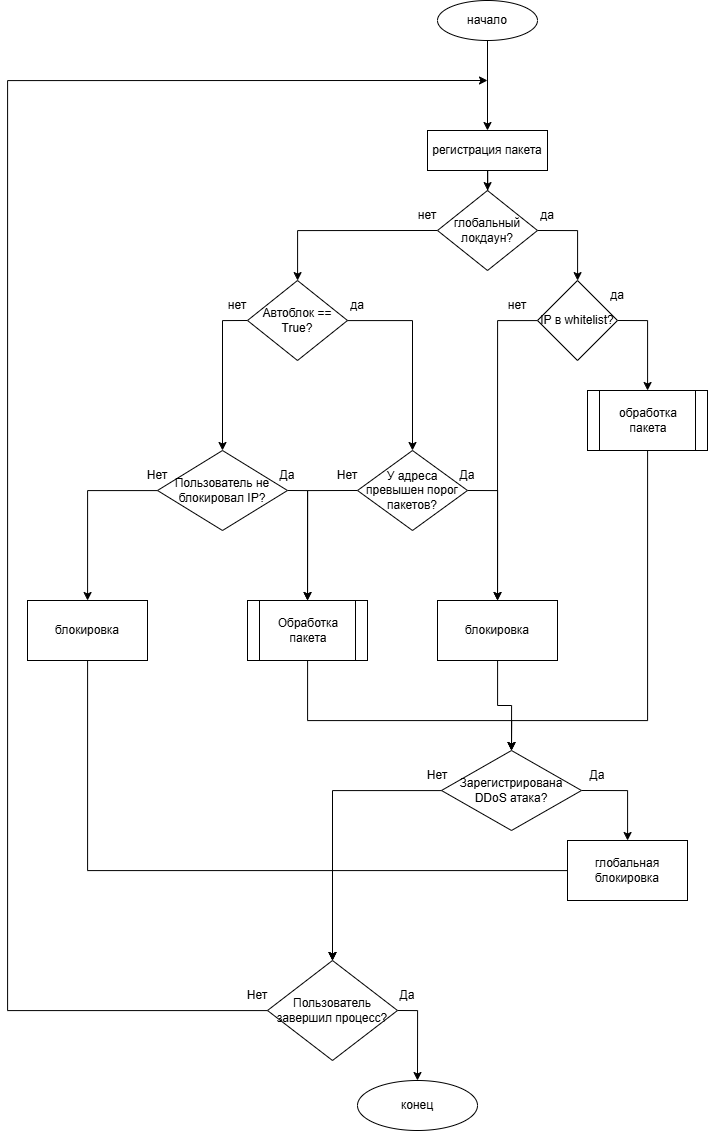
Система применяет меры противодействия автоматически и вручную:

* Персональная блокировка IP на заданный срок/бессрочно;
* Глобальный «локдаун»: блокируются все, кроме whitelist;
* Ручные команды из GUI (Block/Whitelist), авто-блок по порогам.

# **Основные шаги программы**

1. **Запуск**. Программа стартует из GUI или через консольный клиент, где задаются интерфейс, BPF-фильтр и пороги автоблокировок/DDOS. Параметры передаются ядру детекции при нажатии Start либо через аргументы командной строки.
2. **Инициализация**. Ядро создаёт каталог логов, готовит файлы detections.jsonl, commands.jsonl, whitelist.jsonl и загружает белый список. Параллельно запускается фоновый поток, который постоянно читает команды из commands.jsonl.
3. **Сниффинг и предварительные отсеки**. Scapy начинает захват пакетов на заданном интерфейсе с BPF-фильтром, каждый пакет немедленно поступает в обработчик. До детекции выполняются быстрые проверки: активен ли глобальный локдаун и не заблокирован ли источник.
4. **Детекция и журналирование**. К пакету применяются правила: высокая частота от одного источника и обращение к нетипичному порту; при срабатывании формируется причина и событие пишется в detections.jsonl. Одновременно обновляются агрегаты — история прибытий по IP и время последнего появления для подсчёта уникальных источников.
5. **Реакции и интерфейс**. При превышении порогов выполняется автоблокировка IP на заданный срок либо включается глобальный локдаун «все, кроме whitelist». GUI в реальном времени показывает поток детекций и список источников и позволяет оператору вручную отправлять команды Block/Whitelist

# **Блок-схема программы**





# **Описание программы**

Программа написана на языке Python и реализует анализ и обеспечение мер безопасности при работе с сетевым трафиком. Программа включает в себя пользовательский интерфейс, реализацию основного алгоритма анализа и логирование в системный журнал.

**Использованные библиотеки**

**os**: Библиотека для работы с файловой системой и выполнения операций над каталогами и файлами.

**argparse**: Разбор параметров командной строки (интерфейс, BPF-фильтр, пороги автоблокировки и DDOS).

**json**: Сериализация и разбор событий/команд в формате JSON Lines (детекции, мета-события, команды GUI).

**sys**: Применяется для управления процессом завершения программы.

**time**: Метки времени в секундах, измерение интервалов, тайм-ауты и периодический опрос файла команд.

**threading**: Фоновый поток для непрерывного чтения commands.jsonl, чтобы не блокировать обработку пакетов.

**PyQt5**: Библиотека для создания графических интерфейсов (GUI). Позволяет объединять визуальные компоненты с логикой программы.

**Scapy**: Сниффинг пакетов с BPF-фильтрами, доступ к полям IP/TCP/UDP и передача пакетов в правила.

**subprocess**: Старт detector.py как подпроцесса из GUI и чтение его стандартного вывода в реальном времени.**Основные функции программы**

Таблица 1.1. Описание методов detector.py

|  |  |  |
| --- | --- | --- |
| **Название функции** | **Входные переменные** | **описание** |
| **record\_arrival** | **src** | |  | | --- | | Регистрирует пакет: добавляет текущий timestamp в историю и обновляет время последнего появления источника. |  |  | | --- | |  | |
| **count\_recent** | **src, window** | |  | | --- | | Возвращает число пакетов от src за последние window секунд. |  |  | | --- | |  | |
| **emit\_event** | **Pkt, reason, path, extra** | |  |  |  | | --- | --- | --- | | |  | | --- | | Пишет служебное (мета)-событие в stdout и JSONL. |  |  | | --- | |  | |  |  | | --- | |  | |
| **handle\_packet** | **Pkt, jsonl\_path, auto\_block, ab\_treshold, ab\_window, ab\_duration, ddos\_unique\_treshold, ddos\_window\_sec, ddos\_duration** | Основной обработчик пакета: применяет быстрые отсеки (локдаун/персональная блокировка), запускает правила (rules), логирует детекции, обновляет агрегаты, инициирует авто-блок по IP и, при всплеске уникальных источников, включает глобальный локдаун. |
| **poll\_commands** | **Cmd\_path, whitelist\_path** | |  | | --- | | Фоново читает logs/commands.jsonl и исполняет команды: block, whitelist, unwhitelist |  |  | | --- | |  | |
| **main** | **args** | Точка входа: парсит аргументы, создаёт каталог логов, инициализирует/загружает whitelist, запускает поток poll\_commands и scapy.sniff(). |

Таблица 1.2. Описание методов blocker.py

|  |  |  |
| --- | --- | --- |
| **Название функции** | **Входные переменные** | **описание** |
| **add\_whitelist** | **ip** | |  | | --- | | Добавляет IP в whitelist и фиксирует событие в жкрнал. Возвращает True при успехе. |  |  | | --- | |  | |
| **remove\_whitelist** | **ip** | |  | | --- | | Удаляет IP из whitelist и фиксирует в журнал. |  |  | | --- | |  | |
| |  | | --- | | **set\_global\_lockdown** |  |  | | --- | |  | | duration | |  |  |  | | --- | --- | --- | | |  | | --- | | Включает/продлевает глобальный «локдаун» на заданное число секунд: вычисляет end и обновляет global\_lock\_until, если он больше текущего. |  |  | | --- | |  | |  |  | | --- | |  | |
| **block\_ip** | **Ip, duration** | Блокирует IP на время duration. Бессрочная блокировка при duration <=0. |

**Основные переменные**

Таблица 2.1. Описание переменных detector.py

|  |  |
| --- | --- |
| **Имя переменной** | **Функция** |
| **arrival\_history** | |  | | --- | | История временных меток приходов пакетов по каждому источнику для подсчёта активности в окне и автблокировок. |  |  | | --- | |  | |
| **last\_seen\_ts** | |  | | --- | | Время последнего появления каждого источника; используется для подсчёта количества уникальных IP за короткое окно (DDOS-защита) |  |  | | --- | |  | |
| **logs\_dir, whitelist\_file, commands\_file** | Пути к директории с логами, файлу с адресами whitelist и файлу с командами |
| **a.iface, a.bpf, a.block\_threshold, a.block\_window, a.block\_duration, a.ddos\_unique\_threshold, a.ddos\_duration** | |  | | --- | | Настраиваемые параметры – сетевой интерфейс, BPF-фильтр, количество пакетов от одного ip для срабатывания блокировки, ширина временного окна для подсчета пакетов от одного IP, длительность блокировки, порог числа уникальных адресов за окно для глобального локдауна, длительность локдауна. |  |  | | --- | |  | |

# **Контрольный пример**

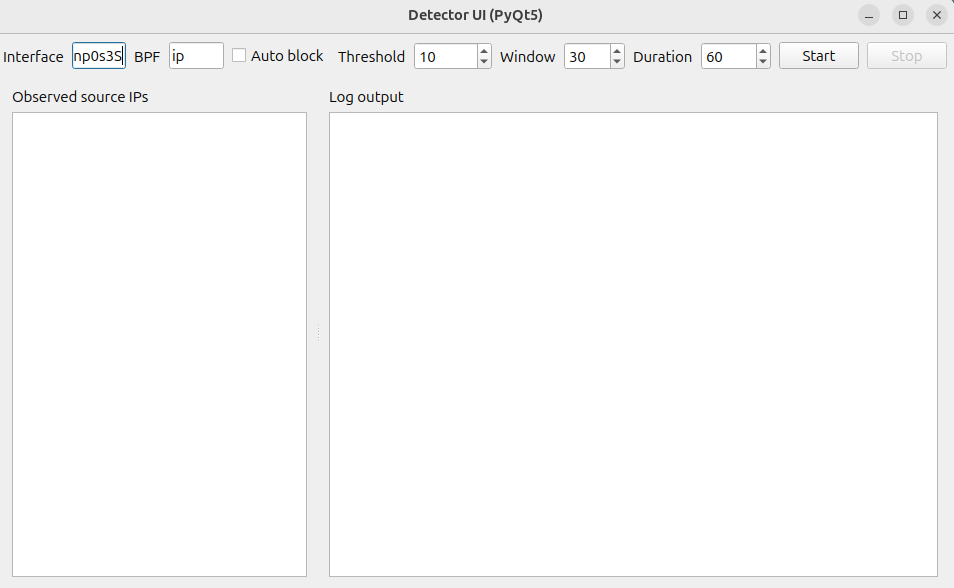


Рисунок 1. Начальное окно программы

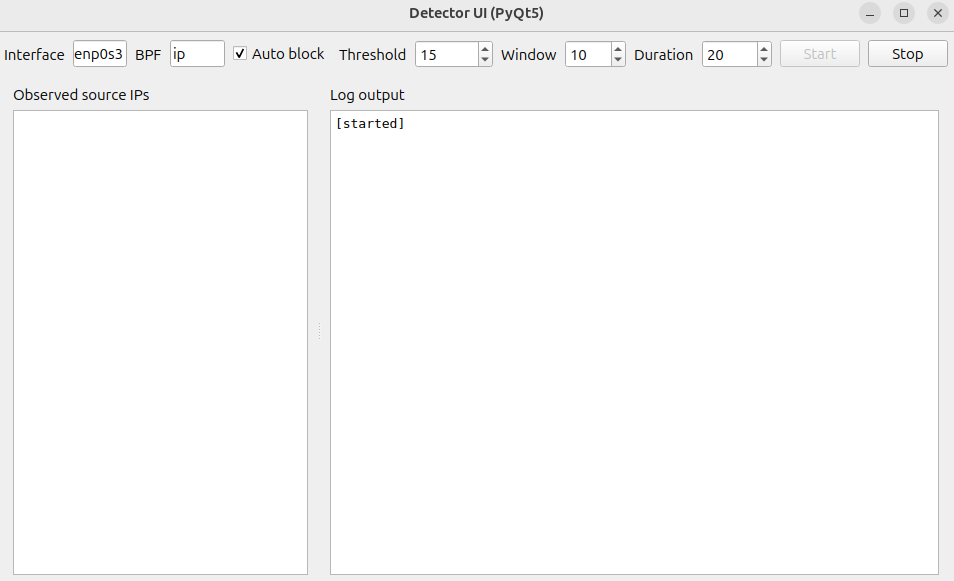


Рисунок 2. Пример запуска детектора с настроенными параметрами

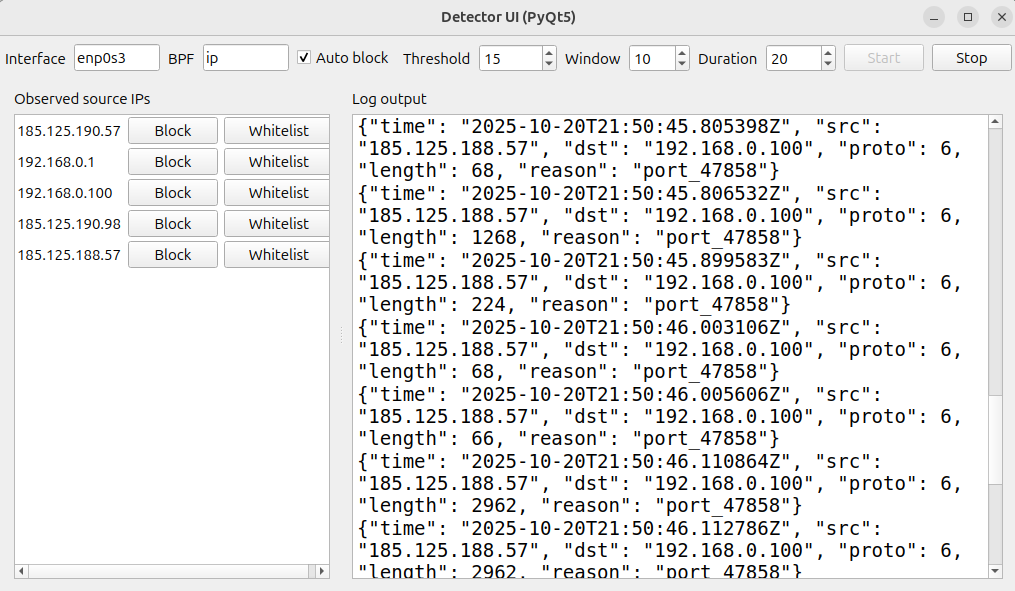


Рисунок 3. Регистрация адресов от которых приходят пакеты и подозрительного поведения

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4. Добавление в whitelist и блокировка выбранных IP

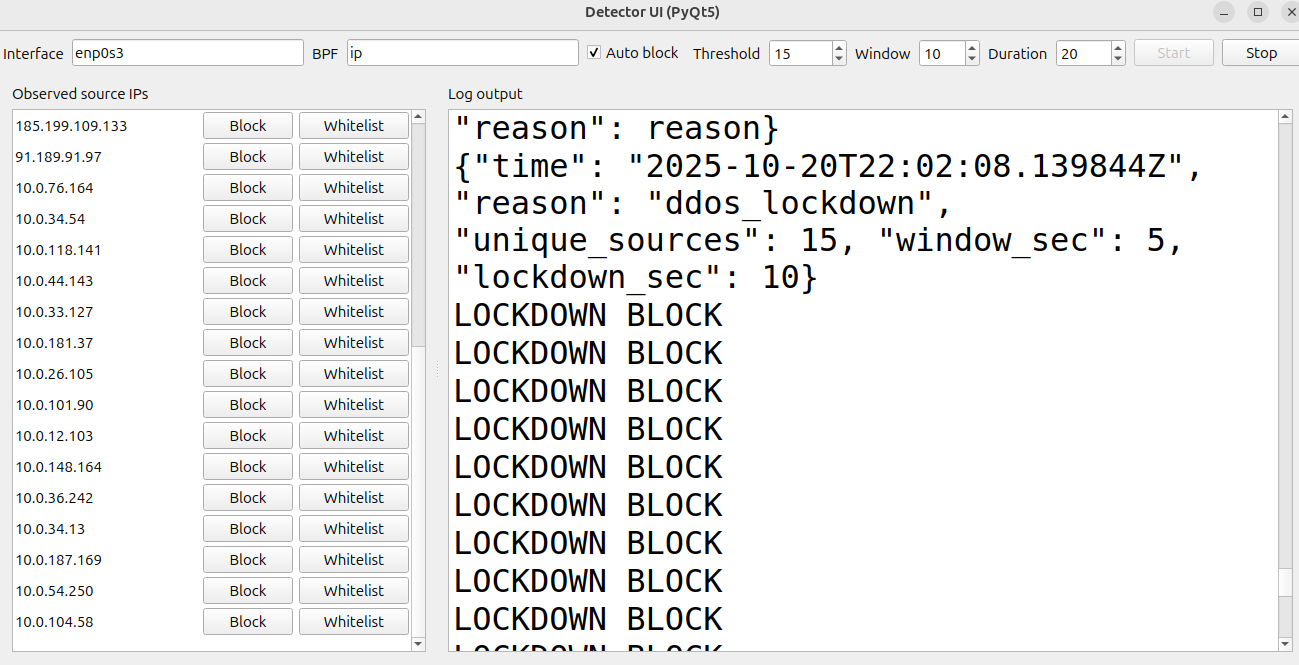


Рисунок 5. Вызов глобального локдауна, как меры защиты от DDoS атаки и соответствующая блокировка входящих пакетов

# **Рекомендации пользователя**

* + 1. **Запуск программы**
* Для запуска используйте командную строку или терминал. Перейдите в каталог с кодом и выполните:  
  sudo python3 detector\_ui.py
* Программа откроет графический интерфейс для работы с детектором.
  + 1. **Настройка фильтров и порогов**
* Задайте BPF-фильтр (например: ip, tcp, tcp port 80, not host 10.0.0.1).
* Первый запуск выполните без авто-блокировки; затем включайте её и подбирайте threshold/window/duration.
* Сразу добавьте доверенные адреса в whitelist.
  1. **Работа с блокировками**
* Управляйте из GUI: **Block** и **Whitelist/Unwhitelist**; все действия журналируются.
* При всплеске уникальных источников включается глобальный локдаун («все, кроме whitelist»).
* Для снятия ограничений удалите IP из блок-листа или временно отключите авто-блок.
  1. **Диагностика и безопасность**
* Просматривайте логи в соответствующем окне
* Если нет событий — проверьте интерфейс, права и BPF-фильтр; при «шумах» повысьте пороги.
* Анализируйте только свой трафик; корректно завершайте работу через **Stop** в GUI или Ctrl+C.

# **Рекомендации для программиста**

* 1. **Настройка окружения**: Рекомендуется Python 3.9+ (Linux предпочтителен), установленный libpcap и пакеты scapy и PyQt5. Для сниффинга выдайте процессу права (запуск от root или Linux-capabilities CAP\_NET\_RAW и CAP\_NET\_ADMIN), убедитесь в доступности нужного сетевого интерфейса, подготовьте каталог logs/ для записи и синхронизируйте системное время (UTC) для корректных меток в JSONL.
  2. **Архитектура и структура программы**: Логику держите модульной: detector.py отвечает за захват/детекцию/журналы, rules.py — за набор правил, blocker.py — за пер-IP и глобальные блокировки и whitelist, detector\_ui.py — за GUI и запуск подпроцесса. Обмен между ядром и интерфейсом организован через поток stdout (JSON-события) и файл команд logs/commands.jsonl, аудит whitelist ведётся в logs/whitelist.jsonl; контракт вызовов к blocker фиксируйте на уровне is\_whitelisted / is\_blocked / block\_ip / set\_global\_lockdown.
  3. **Модификация программы**: Новые детекторы добавляйте в rules.py в виде функций вида rule\_x(pkt, state) → (bool, reason) и подключайте их в handle\_packet, а пороги и окна делайте конфигурируемыми через CLI/GUI или внешний config.json, избегая хардкода. При изменениях схемы логов поддерживайте версионирование и, при необходимости «жёсткой» блокировки, добавьте адаптер к системному фаерволу (iptables/nftables/Windows Firewall) с детальным аудитом.
  4. **Оптимизация, тестирование и эксплуатация**: Сужайте поток трафика BPF-фильтрами, оставляйте store=False и избегайте тяжёлых операций в handle\_packet; структуры состояния ограничивайте по времени (TTL/deque), а запись логов и обработку команд при высокой нагрузке выносите в отдельный поток или очередь и увеличивайте pcap-буфер. Пишите юнит-тесты (pytest) для rules и blocker, проводите e2e-проверки на синтетическом трафике и PCAP, включайте ротацию логов и базовые метрики (pps, число триггеров, активные блокировки) для мониторинга в эксплуатации.

# **Исслндование**

Для симуляции атаки с Windows-хоста протестированы режимы сети ВМ: NAT и Bridged. NAT дал слабую наблюдаемость (пакеты с Windows не видны внутри ВМ), поэтому выбран **Bridged**: адаптер ВМ привязан к физическому интерфейсу хоста, «кабель подключён», при наличии promiscuous-режима — **Allow All**; ВМ получает IP в той же подсети. В детекторе указывался интерфейс (напр., enp0s3) и фильтр ip/icmp.

Изначально трафик генерировался с помощью ping команд для проверки работоспособности сервиса в целом, но позже, для симуляции разных видов атак и защиты было необходимо написать соответствующие скрипты – скрипт для симуляции DDoS атак и атак вида “один IP – много пакетов”. По итогам тестирования были выбраны оптимальные параметры защиты – threshold:10, window:30, duration:60

Из вариантов противодействия DDOS выбран **lockdown + whitelist**: при всплеске числа уникальных источников включается кратковременная блокировка «все, кроме доверенных». Метод быстро гасит шквал и не отключает критичные адреса. Для демонстрации рекомендованы параметры: --ddos-unique-threshold 15 --ddos-window-sec 5 --ddos-duration 10–30 (допустимо уменьшить пороги на стенде с малым числом источников).

# **Вывод**

Реализована модульная система онлайн-детекции сетевых аномалий с автоматическими и ручными мерами реагирования: ядро на Scapy анализирует трафик в реальном времени, правила комбинируют поведенческие и сигнатурные признаки, а подсистема блокировок поддерживает пер-IP блок и краткосрочный **lockdown + whitelist** для подавления всплесков по числу источников. GUI на PyQt5 обеспечивает запуск, наблюдение и управление. Для демонстрации с Windows-хоста оптимален сетевой режим ВМ **Bridged**, что подтверждено практическими тестами (ICMP ping). Логи в формате JSONL упрощают аудит и последующий анализ. Решение легко расширяется новыми правилами и интеграцией с системным фаерволом.

# Список использованной литературы

1. <https://scapy.readthedocs.io>
2. <https://www.tcpdump.org/manpages/pcap-filter.7.html>
3. <https://wiki.archlinux.org/title/iptables>