



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОМУ ПРОЕКТУ*

### *НА ТЕМУ:*

*Метод обнаружения DDoS-атак в сетевом трафике*

Студент ИУ7-74Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата) Н.А. Гурова  
(И.О.Фамилия)

Руководитель курсового проекта

\_\_\_\_\_  
(Подпись, дата) Н.Ю. Рязанова  
(И.О.Фамилия)

2023 г.



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ  
Заведующий кафедрой ИУ7  
(Индекс)  
И.В. Рудаков  
(И.О.Фамилия)  
« \_\_\_\_ » \_\_\_\_\_ 2023 г.

## З А Д А Н И Е на выполнение курсового проекта

по дисциплине Операционные системы

Студент группы ИУ7-74Б

Гурова Наталия Алексеевна  
(Фамилия, имя, отчество)

Тема курсового проекта Метод обнаружения DDoS атак в сетевом трафике

Направленность КП (учебный, исследовательский, практический, производственный, др.)  
учебный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

**Задание** Разработать загружаемый модуль ядра для ОС Linux, осуществляющий обнаружение DDoS атак в сетевом трафике. Также необходимо предоставить пользователю возможность скрывать разработанный загружаемый модуль ядра ОС Linux.

### Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение, список литературы.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.): на защиту работы должна быть предоставлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс.

Дата выдачи задания «\_\_» \_\_\_\_\_ 20\_\_ г.

Руководитель курсового проекта

Н.Ю. Рязанова  
(Подпись, дата) (И.О.Фамилия)

Студент

Н.А. Гурова  
(Подпись, дата) (И.О.Фамилия)

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
1.1 Постановка задачи . . . . .	4
1.2 Сетевая модель OSI . . . . .	4
1.3 Удаленная сетевая атака . . . . .	7
1.4 Атака SYN-flood . . . . .	10
1.5 Выбор отслеживаемых параметров сетевого трафика . . . . .	12
1.6 Программный интерфейс для сетевого взаимодействия в ОС Linux	12
1.7 Фреймворк Netfilter . . . . .	13
1.8 Анализ способа изменения видимости модуля . . . . .	17
1.9 Требования к программному обеспечению для взаимодействия с загружаемым модулем ядра для ОС Linux . . . . .	17
1.10 Выводы . . . . .	19
<b>2 Конструкторский раздел</b>	<b>21</b>
2.1 IDEF0 . . . . .	21
2.2 Структура программного обеспечения . . . . .	22
2.3 Алгоритм инициализации модуля . . . . .	23
2.4 Алгоритм обработки перехваченного сетевого пакета . . . . .	24
2.5 Алгоритм обнаружение атак в исторических данных . . . . .	25
<b>3 Технологический раздел</b>	<b>26</b>
3.1 Выбор средств разработки . . . . .	26
3.2 Сборка и запуск модуля . . . . .	26
3.3 Инициализация модуля . . . . .	26
3.4 Обработка перехваченных сетевых пакетов . . . . .	27
3.5 Обнаружение признаков атак . . . . .	28
3.6 Изменение видимости модуля . . . . .	32
<b>4 Исследовательский раздел</b>	<b>34</b>
4.1 Команды . . . . .	34
4.2 Видимость модуля . . . . .	34

4.3	Вывод статистики по работе модуля . . . . .	34
4.4	Просмотр исторических данных . . . . .	35
4.5	Вывод . . . . .	36
<b>ЗАКЛЮЧЕНИЕ</b>		<b>37</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>		<b>38</b>
<b>ПРИЛОЖЕНИЕ А</b>		<b>39</b>
<b>ПРИЛОЖЕНИЕ Б</b>		<b>41</b>
<b>ПРИЛОЖЕНИЕ В</b>		<b>59</b>
<b>ПРИЛОЖЕНИЕ Г</b>		<b>64</b>

## ВВЕДЕНИЕ

Одной из важнейших задач обеспечения безопасности взаимодействия процессов в распределенных системах является задача контроля входящего трафика на определенный хост.

Возможным методом контроля является динамическое обнаружение атак.

Для решения задачи обнаружения атак во входящем трафике используется специальное программное обеспечение, которое может быть реализовано в виде загружаемого модуля ядра для ОС Linux.

Разработке такого программного обеспечения посвящена данная работа.

# 1 Аналитический раздел

## 1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуль ядра для ОС Linux, осуществляющий динамическое обнаружение атак во входящем сетевом трафике.

Для достижения поставленной задачи необходимо:

- провести анализ принципов работы сетевой подсистемы ОС Linux;
- провести анализ способов перехвата сетевых пакетов;
- провести анализ методов динамического обнаружения атак;
- разработать алгоритм и стороннее ПО для выявления атак в сетевом трафике;
- предусмотреть возможность сокрытия разработанного загружаемого модуля ядра для ОС Linux.

## 1.2 Сетевая модель OSI

В распределённых системах информация передаётся в виде пакетов по протоколам. Передаваемые данные подвергаются процессам инкапсуляции и декапсуляции, в процессе которых каждый пакет проходит все уровни сетевой модели OSI (таблица 1.1).

Таблица 1.1 – Модель OSI

№	Название
7	Прикладной уровень
6	Уровень представления
5	Сеансовый уровень
4	Транспортный уровень
3	Сетевой уровень
2	Канальный уровень
1	Физический уровень

## Internet Protocol

Internet Protocol (IP) — протокол сетевого уровня модели OSI.

Данный протокол объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных между любыми узлами сети через произвольное число промежуточных узлов (маршрутизаторов). Именно поэтому он стал тем протоколом, который объединил отдельные компьютерные сети во всемирную сеть Интернет.

IP не гарантирует надёжной доставки пакета до адресата — в частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (приходят две копии одного пакета), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прийти вовсе.

На рисунке 1.1 представлен заголовок пакета IPv4.

Номер версии	Длина заголовка	Тип сервиса	Длина пакета	
ID пакета			Флаги	Указатель фрагмента
Время жизни пакета		Протокол	Контрольная сумма	
Адрес источника				
Адрес назначения				
Опции				
Заполнитель				

Рисунок 1.1 – Заголовок пакета IP

## Transmission Control Protocol

Transmission Control Protocol (TCP) — протокол транспортного уровня модели OSI.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым (в отличие от UDP) целостность передаваемых данных и уведомление отправителя о результатах передачи.

На рисунке 1.2 представлен заголовок сегмента TCP.

Порт источника		Порт назначения	
Номер последовательности			
Номер подтверждения			
Длина заголовка	Резерв	Флаги	Размер окна
Контрольная сумма		Указатель важности	
Опции			

Рисунок 1.2 – Заголовок пакета TCP

### Флаги:

- **URG** — поле «указатель важности» имеет смысл;
- **ACK** — данный сегмент является подтверждением получения предыдущего;
- **PSH** — нужно передать немедленно;
- **RST** — аварийное закрытие соединения;
- **SYN** — открытие соединения;
- **FIN** — корректное завершение соединения.

**Трехстороннее рукопожатие TCP** — это процесс, который устанавливает соединение между клиентом и сервером. Он состоит из трех этапов, где каждая сторона обменивается определенными пакетами синхронизации и подтверждения, чтобы убедиться, что обе стороны готовы к обмену данными.

1. Клиент отправляет серверу пакет синхронизации (SYN), чтобы начать установку соединения.
2. Сервер получает пакет SYN, затем отправляет пакет синхронизации и подтверждения (SYN-ACK) клиенту, указывая, что он готов к установлению соединения.



3. Клиент затем отправляет серверу пакет подтверждения (ACK), завершая процесс установки соединения.

Принцип выполнения трехстороннего рукопожатия TCP представлен на рисунке 1.3.

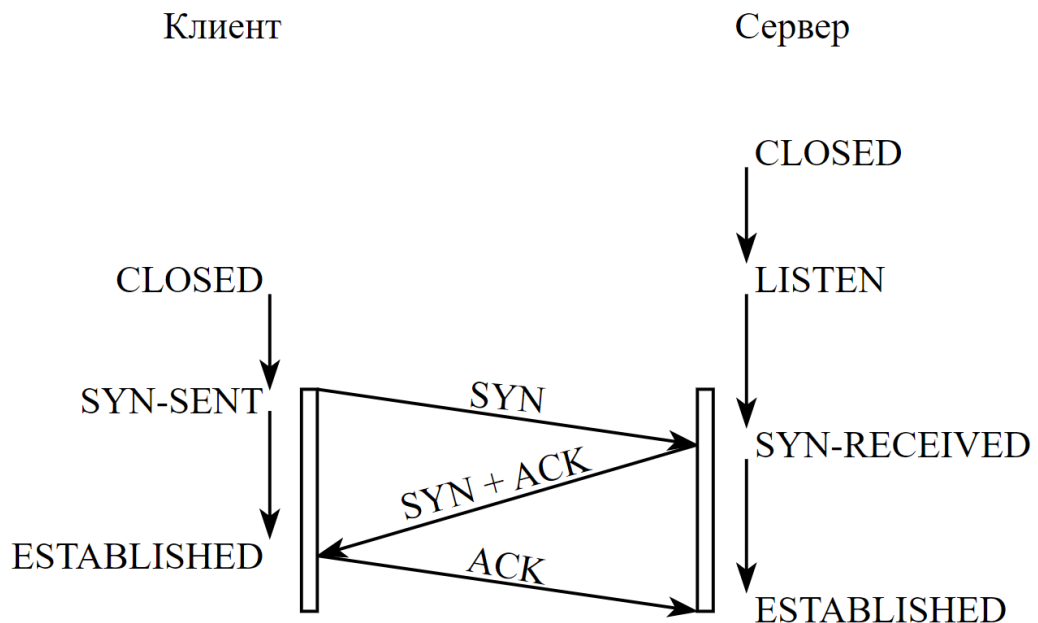


Рисунок 1.3 – Установление соединения TCP

### 1.3 Удаленная сетевая атака

**Удалённая сетевая атака** — информационное разрушающее воздействие на распределённую вычислительную систему, осуществляемое программно по каналам связи.

Выделяют следующие виды удаленных сетевых атак.

#### 1. По характеру воздействия:

- **пассивная атака** — атака не оказывает непосредственное влияние на функционирование системы, но может нарушить ее политику безопасности.
- **активная атака** — атака оказывает непосредственное влияние на работу системы (нарушение работоспособности, искажение или навязывание информации) и нарушающее в ней политику безопасности.

2. По цели воздействия:

- нарушение конфиденциальности информации или параметров системы;
- нарушение целостности информации;
- нарушение работоспособности или доступности компонент системы.

3. По условию начала воздействия:

- **атака по запросу** от атакуемого объекта — атакующая сторона ожидает от потенциальной цели атаки запроса определенного типа, который будет условием начала воздействия;
- **атака по наступлению ожидаемого события** на атакуемом объекте;
- **безусловная атака** — осуществляется независимо от состояния и работоспособности объекта.

4. По наличию обратной связи с атакуемым объектом:

- **с операционной системой** — между атакующим узлом и жертвой существует операционная система, которая позволяет атакующему адекватно реагировать на все изменения, происходящие на стороне жертвы;
- **без операционной системы** или однонаправленная атака.

5. По расположению субъекта атаки относительно жертвы:

- **внутрисегментное** — находится в одной подсети;
- **межсегментное или удаленное** — взаимодействие осуществляется через промежуточные узлы или маршрутизаторы.

6. По уровню модели OSI, на который осуществляется воздействие.

Примерами атак на сетевом уровне являются: перекрытие пакетов, IP Spoofing (подмена IP), ARP Spoofing, затопление ICMP пакетами.

Примеры атак на транспортном уровне: ранняя десинхронизация, локальная буря, затопление SYN пакетами (SYN flood).

## DDoS-атака

**Denial of Service (DoS)** — хакерская атака на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых добросовестные пользователи системы не смогут получить доступ к предоставляемым системным ресурсам (серверам), либо этот доступ будет затруднён. Отказ «вражеской» системы может быть и шагом к овладению системой (если в нештатной ситуации ПО выдаёт какую-либо критическую информацию — например, версию, часть программного кода и так далее). Но чаще это мера экономического давления: потеря простой службы, приносящей доход, счета от провайдера и меры по уходу от атаки ощутимо бьют по карману «цели».

**Distributed Denial of Service** — распределенная DoS-атака. Такая атака проводится в том случае, если требуется вызвать отказ в обслуживании хорошо защищённой крупной компании или правительственной организации.

DDoS-атака обычно состоит из следующих этапов.

1. Злоумышленник сканирует крупную сеть с помощью специально подготовленных сценариев, которые выявляют потенциально слабые узлы.
2. Выбранные узлы подвергаются нападению, и злоумышленник получает на них права администратора.
3. На захваченные узлы устанавливаются программы, которые работают в фоновом режиме. Теперь эти компьютеры называются компьютерами-зомби, их пользователи даже не подозревают, что являются потенциальными участниками DDoS-атаки.
4. Злоумышленник отправляет определенные команды захваченным компьютерам и те, в свою очередь осуществляют коллективную DoS-атаку на целевой компьютер.

На рисунке 1.4 представлен принцип проведения DDoS-атаки.

В настоящее время DoS и DDoS-атаки наиболее популярны, так как позволяют довести до отказа практически любую плохо написанную систему, не оставляя юридически значимых улик.

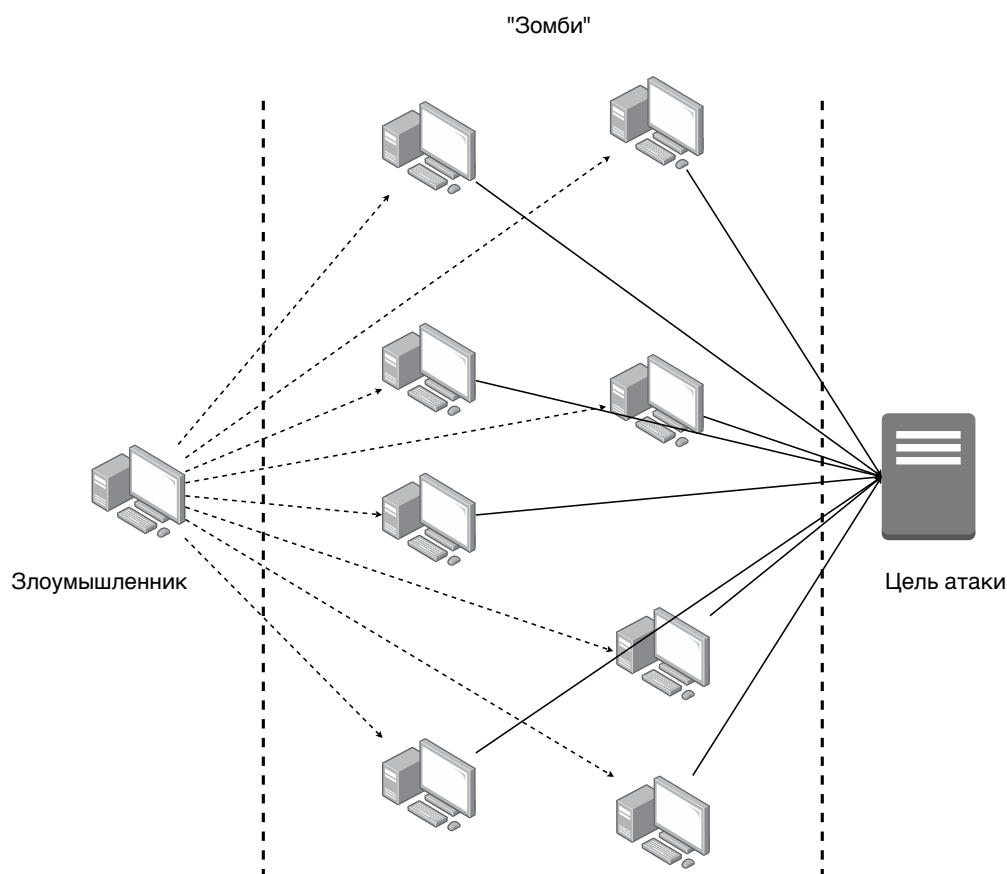


Рисунок 1.4 – DDoS атака

## 1.4 Атака SYN-flood

**Syn-flood атака** — это одна из разновидностей DoS-атак.

Принцип следующий: злоумышленник посылает огромное количество запросов установки соединения на атакуемый сервер. Сервер, видя сегменты с флагом SYN, выделяет необходимые ресурсы для поддержания соединения и отправляет в ответ сегменты с флагами SYN, ACK, переходя в состояние SYN-RECEIVED (такое состояние еще называют полуоткрытым соединением). Злоумышленник, не шлет ответные ACK сегменты, а продолжает бомбардировать сервер SYN-запросами, тем самым вынуждая сервер создавать все больше и больше полуоткрытых соединений. Сервер располагает ограниченными ресурсами, и потому имеет лимит на количество полуоткрытых соединений. Ввиду этой ограниченности, при достижении предельного числа полуоткрытых соединений сервер начинает отклонять новые попытки соединения. Таким образом и достигается отказ в обслуживании (Denial of Service) [1].

## Методы защиты от SYN-flood атак

Для защиты от SYN-flood атак существует два подхода, которые можно комбинировать между собой.

1. Увеличить количество полуоткрытых TCP-соединений и уменьшить время, в котором сокет может пребывать в состоянии SYN-RECEIVED.
2. Использовать SYN-cookie.

Идея SYN-cookie очень проста: когда к нам приходит SYN-запрос мы не создаем новое соединение, а отправляем SYN, ACK ответ клиенту, где в поле Sequence Number кодируем данные о данном соединении. Если мы получаем ACK ответ от клиента, то из поля Acknowledgment Number восстанавливаем данные о соединении. Данный метод хорош тем, что мы более не будем выделять ресурсы, как только к нам придет SYN-запрос. Но есть и очевидный минус: если наш пакет затеряется, или подвергнется искажению, то мы не сможем отправить его повторно, так как информацию о соединении мы условились не сохранять.

## Методы обнаружения SYN-flood атак

Основной признак SYN-flood атак — деградация онлайн-сервиса, а также полная или частичная недоступность IT-систем. Анализ трафика помогает обнаружить атаку и по косвенным признакам, таким как наличие повышенного количества SYN-запросов (SYN-запросы являются нормальной частью установки TCP-соединения, но их избыток может указывать на возможную атаку), подозрительные скачки объёмов трафика с одного IP-адреса (или диапазона IP-адресов).

Кроме анализа трафика для обнаружения SYN-flood атак также используется анализ полуоткрытых соединений и задержек. Эксперименты показывают, что метод исследования задержек способен распознать те полуоткрытые соединения, которые вызваны SYN flood-атакой, и те, которые вызваны другими причинами. Метод получает задержки маршрутизаторов, посылая пакеты, содержащие специально установленные значения TTL в заголовке IP. Результаты зондирования используются затем для надёжного обнаружения SYN-флуда [2].

## 1.5 Выбор отслеживаемых параметров сетевого трафика

Анализируется заголовок сетевого и транспортного уровня, при обработке учитывается информация об IP-адресе отправителя, размере пакета, наличии выставленного флага SYN.

В качестве отслеживаемых параметров выбраны: суммарное количество перехваченных TCP-пакетов, суммарное количество перехваченных TCP-пакетов с флагом  $SYN = 1$ , количество уникальных перехваченных IPv4 адресов, суммарный объем перехваченного трафика (в байтах), суммарное количество перехваченных пакетов, среднее значение суммарного объема перехваченного трафика с одного IPv4 адреса, максимальное значение суммарного объема перехваченного трафика с одного IPv4 адреса, среднее количество перехваченных пакетов с одного IPv4 адреса, максимальное количество перехваченных пакетов с одного IPv4 адреса.

Отслеживаемые параметры содержатся в заголовках IPv4 и TCP, по этой причине для получения отслеживаемых данных необходимо анализировать заголовки IPv4 и TCP (сетевого и транспортного уровня).

## 1.6 Программный интерфейс для сетевого взаимодействия в ОС Linux

Программный интерфейс для сетевого взаимодействия в ОС Linux построен на стеке BSD. В этой подсистеме прием и передача данных на транспортном и сетевом уровнях модели OSI происходят с помощью интерфейса сокетов.

Когда в сетевую подсистему приходит пакет, возникает прерывание. Обработчик прерывания от сетевого адаптера копирует пакет в ядро, где тот ставится в так называемую буферную очередь и ожидает обработки соответствующим потоком ядра, и вызывает соответствующий `softirq`, который будет запущен демоном `ksoftirqd` и закончит обработку данного сетевого пакета (рисунок 1.5) [3].

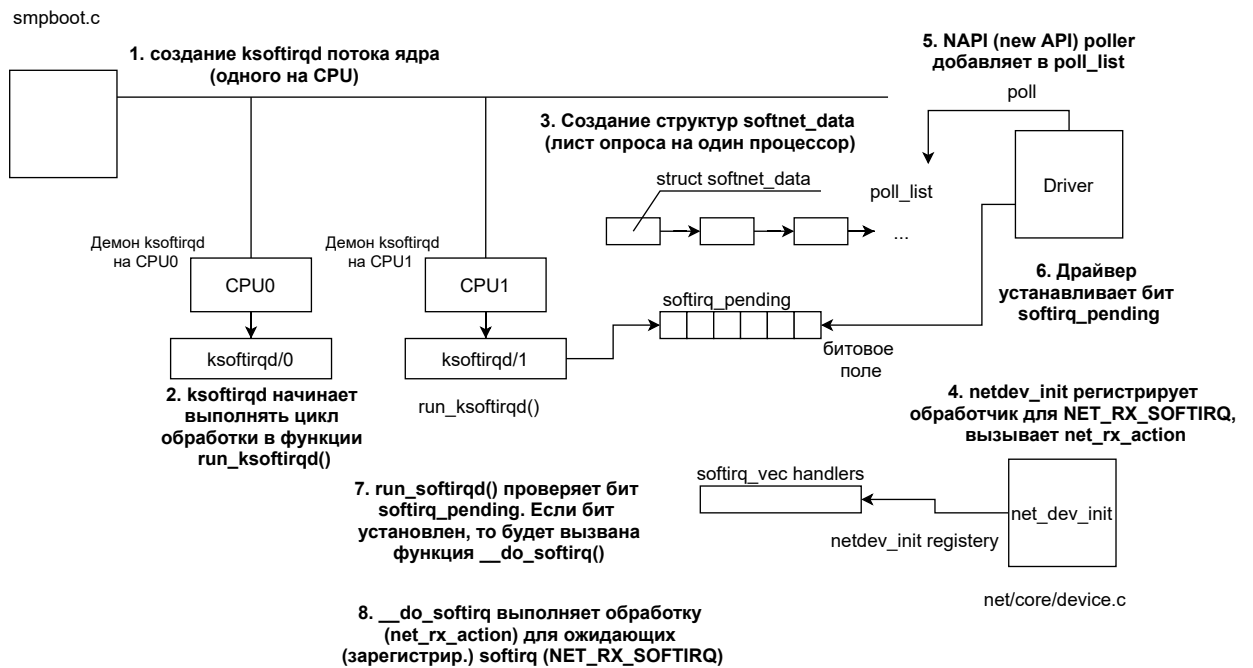


Рисунок 1.5 – Порядок выполняемых действий при приеме сетевого пакета

Для обработки сетевых пакетов, поступающих с высокоскоростных интерфейсов, в Linux был добавлен NAPI (New API). В этом API механизм прерываний сочетается с механизмом опроса, и его основная цель – сократить количество прерываний, генерируемых при получении пакета. В NAPI-совместимых драйверах при поступлении пакета прерывания отключаются, и обработчик только вызывает планировщик `rx_scheduler`, который гарантирует, что в дальнейшем обработка пакета будет выполнена [4].

На рассматриваемых уровнях модели OSI и происходит фильтрация пакетов с учетом информации о протоколе и об IP-адресах и номерах портов источника и назначения. Широко распространённым средством фильтрации сетевых пакетов является библиотека Netfilter.

## 1.7 Фреймворк Netfilter

Netfilter - это фреймворк, который предоставляется ядром ОС Linux, предоставляющая функционал для фильтрации и перенаправления пакетов [5]. Netfilter представляет собой набор перехватчиков внутри ядра, позволяющий модулям ядра регистрировать функции обратного вызова в сетевом стеке. Эти функции вызываются для каждого пакета, который удовлетворяет соответствующему правилу перехвата [6].

Netfilter позволяет перехватить пакет в любой из пяти стандартных

точек, через которые он проходит (рисунок 1.6):

1. `NF_INET_PRE_ROUTING` – все входящие пакеты;
2. `NF_INET_LOCAL_IN` – входящие пакеты, предназначенные для локального процесса;
3. `NF_INET_FORWARD` – транзитные пакеты (предназначенные для другого интерфейса);
4. `NF_INET_LOCAL_OUT` – исходящие пакеты, сформированные локальными процессами;
5. `NF_INET_POST_ROUTING` – все исходящие пакеты (поступившие из точки `NF_INET_FORWARD` или сформированные локальными процессами).

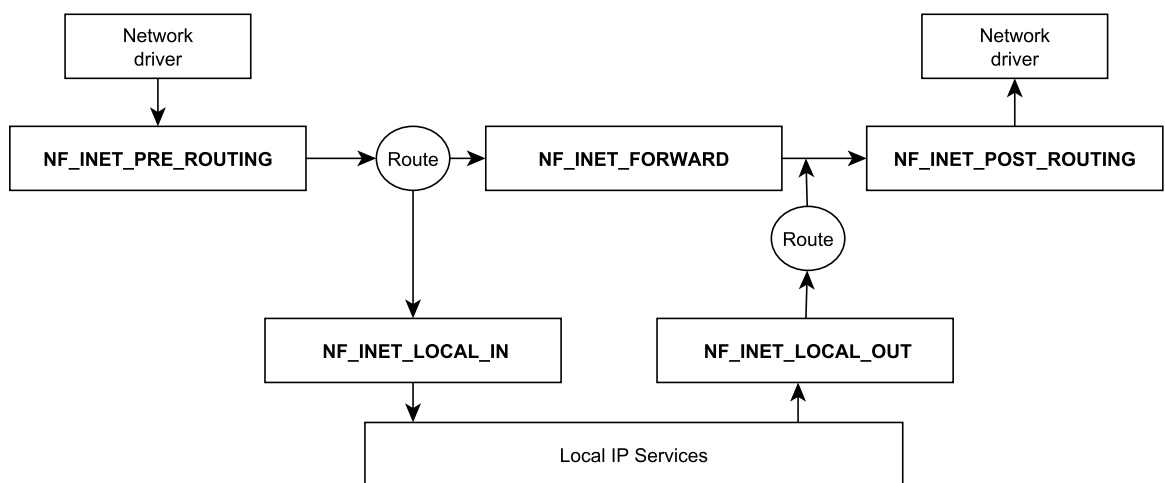


Рисунок 1.6 – Точки, через которые проходит сетевой пакет

## Функции перехвата

Чтобы перехватить пакет в одной из перечисленных ранее точек, к ней необходимо подключить функцию перехвата (hook, ловушку). Для этого сначала необходимо заполнить структуру `nf_hook_ops`, основные поля которой приведены в листинге 1.1.



Листинг 1.1 – Структура struct nf\_hook\_ops

```
1 struct nf_hook_ops {
2     nf_hookfn      *hook;
3     ...
4     u8             pf;
5     ...
6     unsigned int    hooknum;
7     int            priority;
8 };
```

Основные поля структуры:

- hook – функция, которая будет вызвана для обработки пакета и принятия решения: отбросить или принять пакет;
- pf – семейство протоколов (PF\_INET для IPv4);
- hooknum – точка подключения функции;
- priority – приоритет (вводится, чтобы установить порядок вызова ловушек, подключенных к одной точке).

После заполнения структуры nf\_hook\_ops функцию перехвата необходимо зарегистрировать, вызвав функцию nf\_register\_net\_hook. Для удаления ловушки необходимо вызвать функцию nf\_unregister\_net\_hook. Прототипы этих функций приведены в листинге 1.2.

Листинг 1.2 – Функции для регистрации и удаления функций перехвата

```
1 int nf_register_net_hook(struct net *net, const struct
   nf_hook_ops *ops);
2
3 void nf_unregister_net_hook(struct net *net, const struct
   nf_hook_ops *ops);
```

Прототип функции перехвата приведен в листинге 1.3.

Листинг 1.3 – Прототип функции-ловушки

```
1 typedef unsigned int nf_hookfn(void *priv, struct sk_buff *skb,
   const struct nf_hook_state *state);
```

Аргументы функции:

- priv – код одной из пяти точек подключения ловушки;

- `skb` – указатель на структуру `sk_buff`, содержащую информацию о сетевом пакете;
- `state` – указатель на структуру `nf_hook_state`, содержащую информацию, связанную с перехватом пакета (интерфейс ввода/вывода, приоритет и т. д.).

Все сетевые пакеты в ядре представляются структурой `struct sk_buff`, поля которой приведены в листинге 1.4.

Листинг 1.4 – Структура `struct sk_buff`

```

1  struct sk_buff {
2      /* временная метка */
3      ktime_t      tstamp;
4
5      /* указатель на сокет, который отправил или получил этот пак
        ет */
6      struct sock      *sk;
7
8      /* указатель на устройство, с которого получен или которому
        будет отправлен пакет */
9      struct net_device  *dev;
10
11     /* L3 протокол */
12     __be16      protocol;
13
14     /* смещения заголовков относительно head */
15     __u16      transport_header;
16     __u16      network_header;
17     __u16      mac_header;
18
19     /* указатели на конец и начало данных */
20     sk_buff_data_t      tail;
21     sk_buff_data_t      end;
22
23     /*
24     head — указатель на начало буфера выделенного под данные
25     data — указатель на начало данных.
26     */
27     unsigned char      *head, *data;
28
29     // счетчик ссылок

```

```
30         atomic_t      users ;
31     };
```

## 1.8 Анализ способа изменения видимости модуля

Для того, чтобы заинтересованный пользователь не смог удалить средство контроля трафика, его необходимо сделать невидимым. Удаляя соответствующий элемент из связного списка модулей, он становится скрытым, и система не может предоставить информацию о данном загружаемом модуле.

Для взаимодействия со списком модулей необходимо использовать структуру **struct list\_head** и функции **list\_add**, **list\_del**. Перед удалением модуля из списка, следует сохранить его указатель для обеспечения возможности восстановить его видимость.

Структура **struct module** (Листинг 1.5), описывающая модуль, предоставляет доступ к связному списку, где поле **list** – его элемент.

Листинг 1.5 – struct module

```
1 struct module
2 {
3     enum module_state state;
4     struct list_head list; /* Member of list of modules */
5     char name[MODULE_NAME_LEN]; /* Unique handle for this module */
6     ...
7 };
```

## 1.9 Требования к программному обеспечению для взаимодействия с загружаемым модулем ядра для ОС Linux

ПО для взаимодействия с разрабатываемым загружаемым модулем ядра для ОС Linux создается для исключения возможности попадания некорректных данных в пространство ядра и облегчения взаимодействия пользователя с загружаемым модулем ядра.

Задача данного ПО — передать данные полученные от пользователя, в фиксированном формате, в пространство ядра.

Для взаимодействия с загружаемым модулем ядра для ОС Linux через стороннее ПО необходимо реализовать возможность:

- демонстрировать статус работы загружаемого модуля ядра для ОС Linux;
- демонстрировать анализируемые модулем параметры трафика;
- скрывать загружаемым модуль ядра для ОС Linux;
- восстанавливать видимость загружаемого модуля ядра для ОС Linux;

Далее рассмотрены разработанные для описанного ПО типы данных.

В листинге 1.6 приведен тип данных, который описывает выполняемое пользователем действие.

Листинг 1.6 – command\_type

```
1 enum command_type {
2     UNDEFINED = 0,
3     HIDE = 1,
4     UNHIDE = 2,
5 };
```

В листинге 1.7 приведен тип данных, который описывает информацию, которая передается в пространство ядра.

Листинг 1.7 – command

```
1 struct command
2 {
3     enum command_type type;
4 };
```

В листинге 1.8 приведен тип данных, который описывает отслеживаемый параметр.

Листинг 1.8 – l2\_property

```
1 enum l2_property {
2     L2_PROPERTY_UNDEFINED = 0,
3     L2_PROPERTY_UNIQUE_SADDRS_COUNT = 1,
4     L2_PROPERTY_CATCHED_PACKETS_COUNT = 2,
5     L2_PROPERTY_TOT_LEN = 3,
6     L2_PROPERTY_AVG_CATCHED_PACKETS = 4,
7     L2_PROPERTY_MAX_CATCHED_PACKETS = 5,
8     L2_PROPERTY_AVG_LEN = 6,
9     L2_PROPERTY_MAX_LEN = 7,
10    L2_PROPERTY_TOT_TCP = 8,
```

```

11     L2_PROPERTY_TOT_SYN = 9,
12 };

```

В листинге 1.9 приведен тип данных, который описывает отслеживаемые параметры трафика.

Листинг 1.9 – l2\_data\_slice

```

1 struct l2_data_slice
2 {
3     uint64_t from_time;
4     uint32_t unique_saddr_count;
5     uint32_t caught_packets_count;
6     uint32_t tot_len;
7     uint32_t avg_caught_packets;
8     uint32_t max_caught_packets;
9     uint32_t avg_len;
10    uint32_t max_len;
11    int crit_behaviour;
12    enum l2_property property;
13    enum l2_crit_behaviour_type type;
14    uint32_t tot_tcp;
15    uint32_t tot_syn;
16 };

```

В листинге 1.10 приведен тип данных, который описывает статус работы загружаемого модуля ядра для ОС Linux.

Листинг 1.10 – module\_stats

```

1 struct module_stats
2 {
3     uint32_t history_length;
4     uint32_t current_history_length;
5     uint64_t current_period_ns;
6     uint32_t crit_behaviour_count;
7     uint64_t first_crit_behaviour_ns;
8     uint64_t last_crit_behaviour_ns;
9 };

```

## 1.10 Выводы

В результате проведенного анализа был выбран способ перехвата входящих пакетов — путем регистрации функций перехвата с использованием

фреймворка Netfilter. В качестве точки перехвата предлагается использовать точку, в которую проходят пакеты адресованные локальному процессу (NF\_INET\_LOCAL\_IN).

Были рассмотрены структуры и функции ядра, предоставляющие информацию о сетевых пакетах, был рассмотрен метод сокрытия загружаемого модуля ядра ОС Linux.

Отслеживаемые параметры представляют агрегированную информацию за фиксированный промежуток времени, которая регулярно сбрасывается.

В качестве отслеживаемых параметров выбраны: суммарное количество перехваченных TCP-пакетов, суммарное количество перехваченных TCP-пакетов с флагом  $\text{SYN} = 1$ , количество уникальных перехваченных IPv4 адресов, суммарный объем перехваченного трафика (в байтах), суммарное количество перехваченных пакетов, среднее значение суммарного объема перехваченного трафика с одного IPv4 адреса, максимальное значение суммарного объема перехваченного трафика с одного IPv4 адреса, среднее количество перехваченных пакетов с одного IPv4 адреса, максимальное количество перехваченных пакетов с одного IPv4 адреса.

Для анализа исторических данных использовался поиск средних значений анализируемых параметров [7]. Такой подход позволяет корректно обрабатывать глобальные изменения в поведении анализируемого параметров.

## 2 Конструкторский раздел

### 2.1 IDEF0

На рисунках 2.1-2.2 представлена IDEF0-диаграмма, описывающая работу разрабатываемого программного обеспечения.

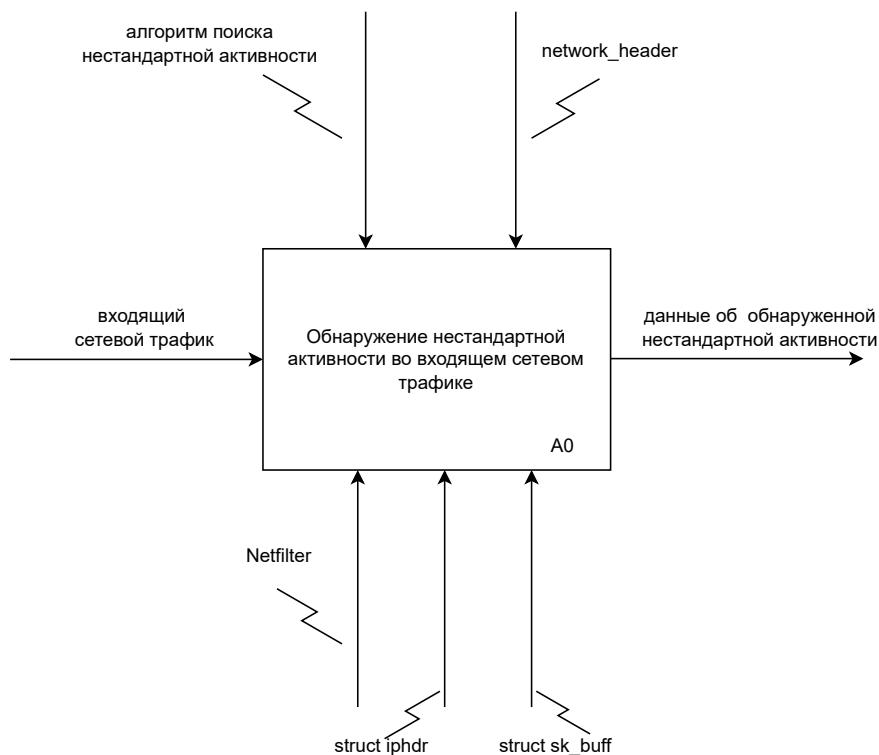


Рисунок 2.1 – Отслеживание сетевого трафика, верхний уровень

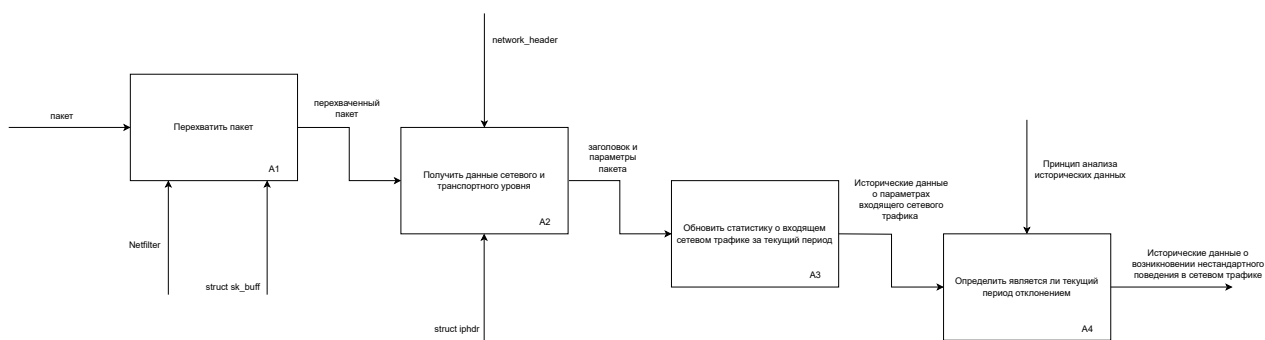


Рисунок 2.2 – Отслеживание сетевого трафика, нижний уровень

## 2.2 Структура программного обеспечения

Разрабатываемое ПО должно быть реализован в виде загружаемого модуля ядра и работать в режиме ядра. Однако пользователь должен иметь возможность управлять работой ПО (скрывать модуль и восстанавливать его видимость), для чего необходимо разработать отдельное приложение, которое будет выполняться в режиме пользователя.

Структура программного обеспечения приведена на рисунке 2.3.

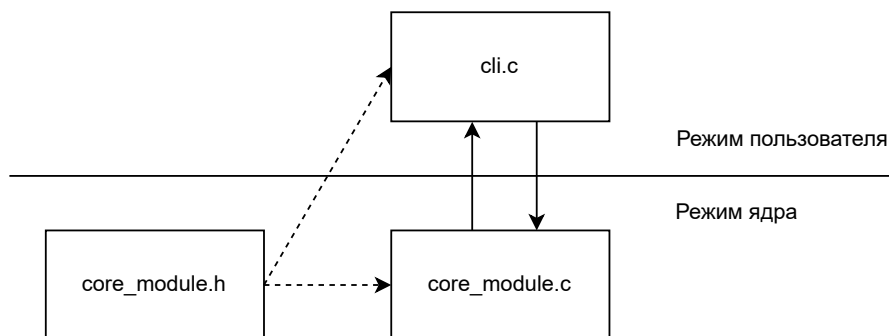


Рисунок 2.3 – Структура программного обеспечения



## 2.3 Алгоритм инициализации модуля

На рисунке 2.4 приведен алгоритм инициализации модуля.

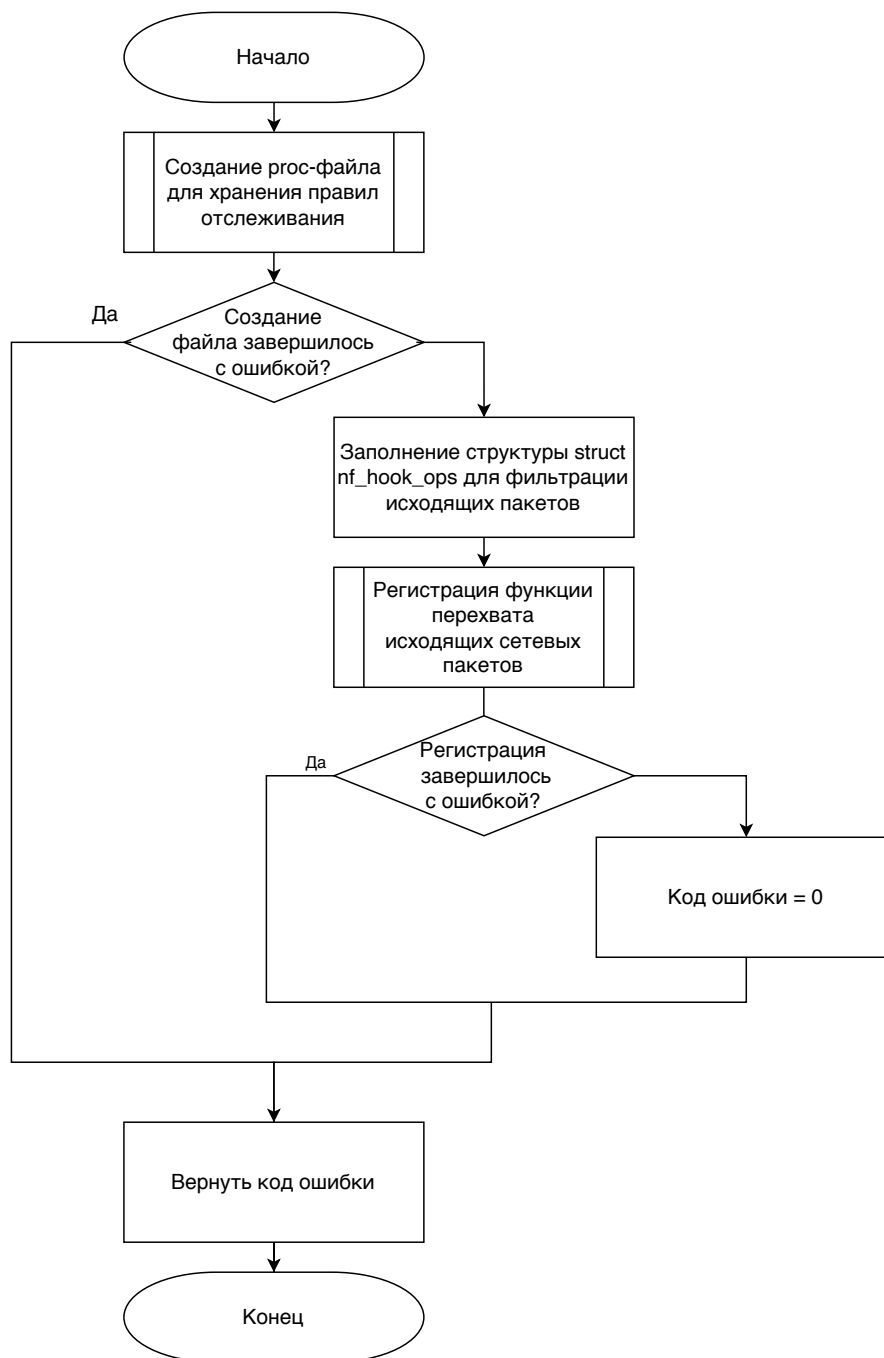


Рисунок 2.4 – Алгоритм инициализации модуля

## 2.4 Алгоритм обработки перехваченного сетевого пакета

На рисунке 2.5 приведен алгоритм обработки перехваченного сетевого пакета.

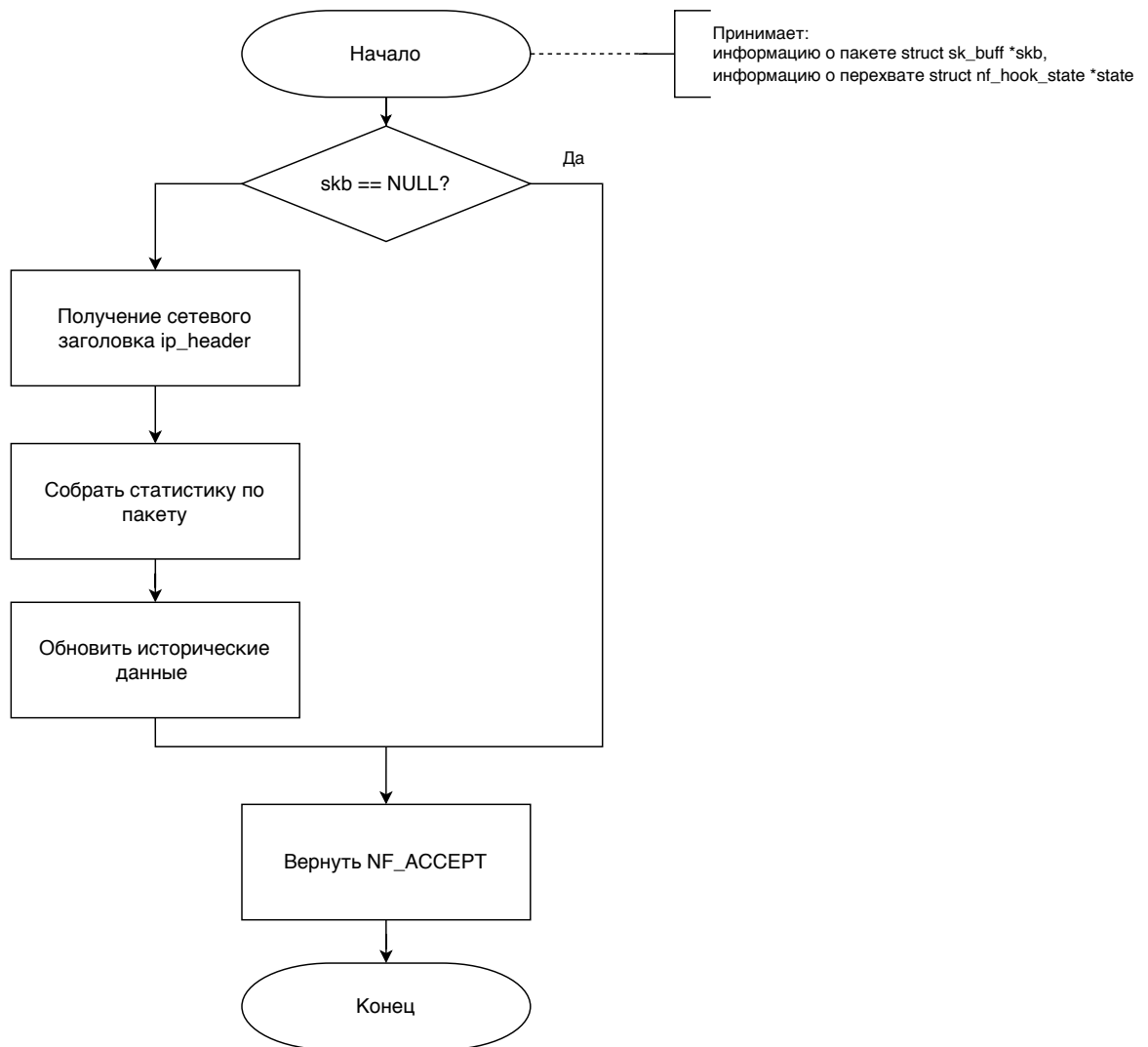


Рисунок 2.5 – Алгоритм обработки перехваченного сетевого пакета

## 2.5 Алгоритм обнаружение атак в исторических данных

На рисунке 2.6 приведен алгоритм обнаружение аномалий в исторических данных для определенного параметра.

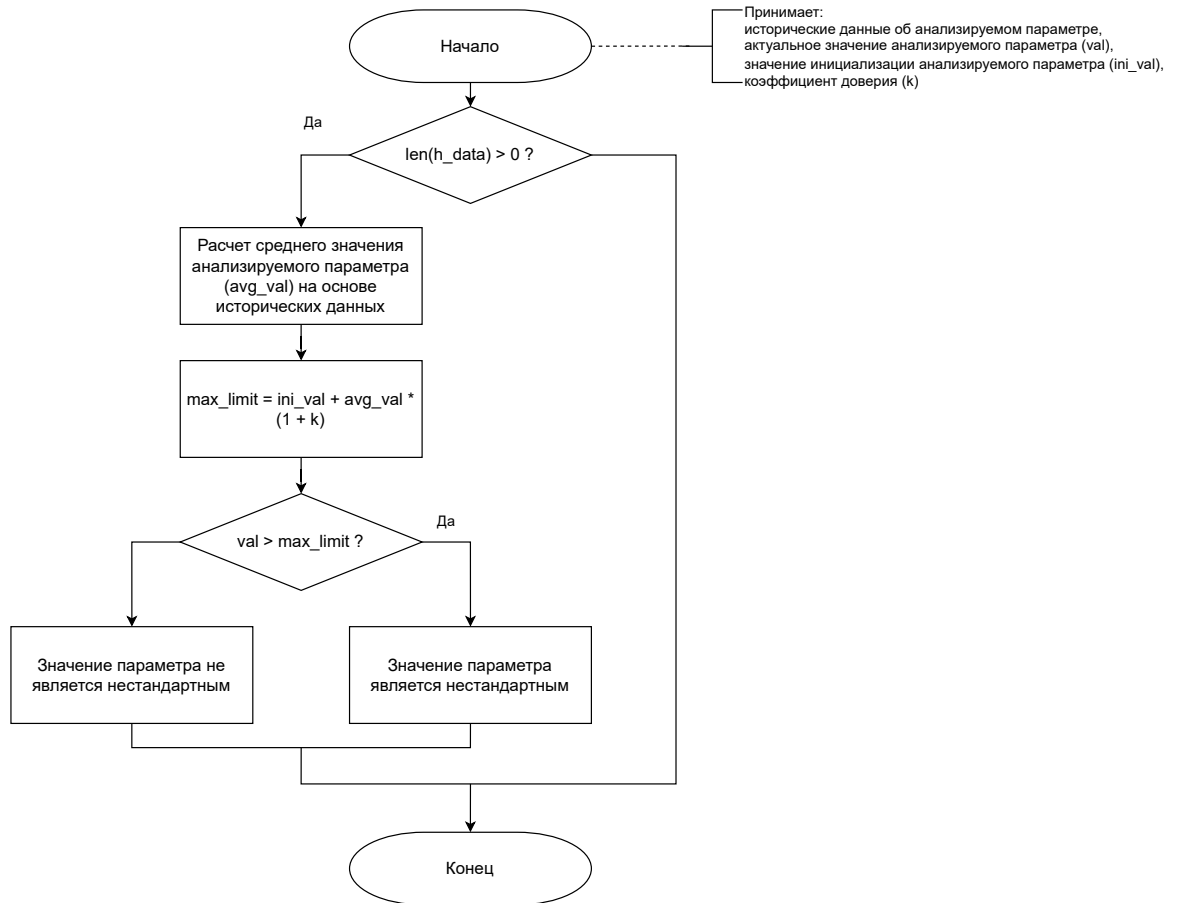


Рисунок 2.6 – Алгоритм обнаружения атак в исторических данных

## 3 Технологический раздел

### 3.1 Выбор средств разработки

В качестве языка программирования для реализации поставленной задачи был выбран язык Си. Для сборки модуля использовалась утилита `make`. В качестве среды разработки был выбран Qt Creator[8], так как он кросс-платформенный, бесплатный и использовался в курсе программирования ранее.

### 3.2 Сборка и запуск модуля

Сборка модуля осуществляется командой `make`. На листинге 3.1 приведено содержимое `Makefile`.

Листинг 3.1 – `Makefile`

```
1 obj-m += core_module.o
2
3 all: interface.o core_module.o
4
5 interface.o: interface.c module.h
6     gcc -o interface.o interface.c
7
8 core_module.o: core_module.c
9     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
10
11 clean:
12     rm -rf fw *.o
13     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Для того, чтобы загрузить модуль, нужно воспользоваться командой `sudo insmod core_module.ko`, для того, чтобы выгрузить – `sudo rmmod core_module`.

### 3.3 Инициализация модуля

В листинге 3.2 приведена реализация функции инициализации модуля.

Листинг 3.2 – Функция инициализации модуля

```
1 static int __init my_module_init(void) {
2     int rc = 0;
3 }
```

```

4     nf_register_net_hook(&init_net, &module_hook_ops);
5
6     rc = misc_register(&dev_l2_data);
7     if (rc)
8     {
9         printk(MODULE_DMESG_PREFIX "[ERROR] registration was
          failed");
10        return rc;
11    }
12
13    rc = misc_register(&dev_stats);
14    if (rc)
15    {
16        printk(MODULE_DMESG_PREFIX "[ERROR] registration was
          failed");
17        return rc;
18    }
19
20    rc = misc_register(&dev_ctrl);
21    if (rc)
22    {
23        printk(MODULE_DMESG_PREFIX "[ERROR] registration was
          failed");
24        return rc;
25    }
26
27    printk(MODULE_DMESG_PREFIX "module was loaded");
28
29    return 0;
30 }

```

### 3.4 Обработка перехваченных сетевых пакетов

В листинге 3.3 приведена реализация функции обработки перехваченных сетевых пакетов.

Листинг 3.3 – Функции обработки перехваченных сетевых пакетов

```

1 static unsigned int catch_traffic(void *priv, struct sk_buff *skb,
   const struct nf_hook_state *state)
2 {
3     struct iphdr *iph; /* An IPv4 packet header */
4     uint32_t saddr;

```

```

5     uint32_t tot_len;
6
7     if (!skb) return NF_ACCEPT;
8
9     iph = (struct iphdr *)skb_network_header(skb);
10    if (iph == NULL) return NF_ACCEPT;
11
12    saddr = iph->saddr;
13    tot_len = iph->tot_len;
14
15    add_packet_info(saddr, tot_len);
16
17    return NF_ACCEPT;
18 }

```

### 3.5 Обнаружение признаков атак

В листинге 3.4 приведена реализация функции обнаружения нестандартной активности во входящем сетевом трафике.

Листинг 3.4 – Функции обнаружения нестандартного активности во входящем сетевом трафике

```

1 void inspect_last(void) {
2     if (second_layer_table_next_index < 2) {
3         return;
4     }
5
6     struct second_layer_schema avg_values = {
7         .unique_saddr_count = 0,
8         .caught_packets_count = 0,
9         .tot_len = 0,
10        .avg_caught_packets = 0,
11        .max_caught_packets = 0,
12        .avg_len = 0,
13        .max_len = 0,
14        .tot_tcp = 0,
15        .tot_syn = 0,
16    };
17
18    for (int i = 0; i < second_layer_table_next_index - 1; i++) {
19        avg_values.unique_saddr_count +=
            second_layer_table[i].unique_saddr_count;

```

```

20     avg_values.catched_packets_count +=
        second_layer_table[i].catched_packets_count;
21     avg_values.tot_len += second_layer_table[i].tot_len;
22     avg_values.avg_catched_packets +=
        second_layer_table[i].avg_catched_packets;
23     avg_values.max_catched_packets +=
        second_layer_table[i].max_catched_packets;
24     avg_values.avg_len += second_layer_table[i].avg_len;
25     avg_values.max_len += second_layer_table[i].max_len;
26     avg_values.tot_tcp += second_layer_table[i].tot_tcp;
27     avg_values.tot_syn += second_layer_table[i].tot_syn;
28 }
29
30 avg_values.unique_saddr_count /= (second_layer_table_next_index
    - 1);
31 avg_values.catched_packets_count /=
    (second_layer_table_next_index - 1);
32 avg_values.tot_len /= (second_layer_table_next_index - 1);
33 avg_values.avg_catched_packets /=
    (second_layer_table_next_index - 1);
34 avg_values.max_catched_packets /=
    (second_layer_table_next_index - 1);
35 avg_values.avg_len /= (second_layer_table_next_index - 1);
36 avg_values.max_len /= (second_layer_table_next_index - 1);
37 avg_values.tot_tcp /= (second_layer_table_next_index - 1);
38 avg_values.tot_syn /= (second_layer_table_next_index - 1);
39
40 struct second_layer_schema last =
    second_layer_table[second_layer_table_next_index - 1];
41
42 uint64_t limit;
43
44 limit = avg_values.unique_saddr_count * (100 + eps_percent) /
    100 + initial_vec.unique_saddr_count;
45 if (last.unique_saddr_count > limit) {
46     second_layer_table[second_layer_table_next_index -
        1].crit_behaviour = 1;
47     second_layer_table[second_layer_table_next_index -
        1].property =
        CRIT_BEHAVIOUR_PROPERTY_UNIQUE_SADDRS_COUNT;
48     second_layer_table[second_layer_table_next_index - 1].type

```

```

    = CRIT_BEHAVIOUR_TYPE_INCREASE;
49     return;
50 }
51
52     limit = avg_values.catched_packets_count * (100 + eps_percent)
    / 100 + initial_vec.catched_packets_count;
53     if (last.catched_packets_count > limit) {
54         second_layer_table[second_layer_table_next_index -
    1].crit_behaviour = 1;
55         second_layer_table[second_layer_table_next_index -
    1].property =
    CRIT_BEHAVIOUR_PROPERTY_CATCHED_PACKETS_COUNT;
56         second_layer_table[second_layer_table_next_index - 1].type
    = CRIT_BEHAVIOUR_TYPE_INCREASE;
57         return;
58     }
59
60     limit = avg_values.tot_len * (100 + eps_percent) / 100 +
    initial_vec.tot_len;
61     if (last.tot_len > limit) {
62         second_layer_table[second_layer_table_next_index -
    1].crit_behaviour = 1;
63         second_layer_table[second_layer_table_next_index -
    1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_LEN;
64         second_layer_table[second_layer_table_next_index - 1].type
    = CRIT_BEHAVIOUR_TYPE_INCREASE;
65         return;
66     }
67
68     limit = avg_values.avg_catched_packets * (100 + eps_percent) /
    100 + initial_vec.avg_catched_packets;
69     if (last.avg_catched_packets > limit) {
70         second_layer_table[second_layer_table_next_index -
    1].crit_behaviour = 1;
71         second_layer_table[second_layer_table_next_index -
    1].property =
    CRIT_BEHAVIOUR_PROPERTY_AVG_CATCHED_PACKETS;
72         second_layer_table[second_layer_table_next_index - 1].type
    = CRIT_BEHAVIOUR_TYPE_INCREASE;
73         return;
74     }

```



```

75
76     limit = avg_values.max_catched_packets * (100 + eps_percent) /
100 + initial_vec.max_catched_packets;
77     if (last.max_catched_packets > limit) {
78         second_layer_table[second_layer_table_next_index -
1] .crit_behaviour = 1;
79         second_layer_table[second_layer_table_next_index -
1] .property =
            CRIT_BEHAVIOUR_PROPERTY_MAX_CATCHED_PACKETS;
80         second_layer_table[second_layer_table_next_index - 1].type
            = CRIT_BEHAVIOUR_TYPE_INCREASE;
81         return;
82     }
83
84     limit = avg_values.avg_len * (100 + eps_percent) / 100 +
            initial_vec.avg_len;
85     if (last.avg_len > limit) {
86         second_layer_table[second_layer_table_next_index -
1] .crit_behaviour = 1;
87         second_layer_table[second_layer_table_next_index -
1] .property = CRIT_BEHAVIOUR_PROPERTY_AVG_LEN;
88         second_layer_table[second_layer_table_next_index - 1].type
            = CRIT_BEHAVIOUR_TYPE_INCREASE;
89         return;
90     }
91
92     limit = avg_values.max_len * (100 + eps_percent) / 100 +
            initial_vec.max_len;
93     if (last.max_len > limit) {
94         second_layer_table[second_layer_table_next_index -
1] .crit_behaviour = 1;
95         second_layer_table[second_layer_table_next_index -
1] .property = CRIT_BEHAVIOUR_PROPERTY_MAX_LEN;
96         second_layer_table[second_layer_table_next_index - 1].type
            = CRIT_BEHAVIOUR_TYPE_INCREASE;
97         return;
98     }
99
100    limit = avg_values.tot_tcp * (100 + eps_percent) / 100 +
            initial_vec.tot_tcp;
101    if (last.tot_tcp > limit) {

```

```

102     second_layer_table[second_layer_table_next_index -
103         1].crit_behaviour = 1;
104     second_layer_table[second_layer_table_next_index -
105         1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_TCP;
106     second_layer_table[second_layer_table_next_index - 1].type
107         = CRIT_BEHAVIOUR_TYPE_INCREASE;
108     return;
109 }
110
111 limit = avg_values.tot_syn * (100 + eps_percent) / 100 +
112     initial_vec.tot_syn;
113 if (last.tot_syn > limit) {
114     second_layer_table[second_layer_table_next_index -
115         1].crit_behaviour = 1;
116     second_layer_table[second_layer_table_next_index -
117         1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_SYN;
118     second_layer_table[second_layer_table_next_index - 1].type
119         = CRIT_BEHAVIOUR_TYPE_INCREASE;
120     return;
121 }
122 }

```

### 3.6 Изменение видимости модуля

В листинге 3.5 приведена реализация функции сокрытия разработанного загружаемого модуля ядра ОС Linux.

Листинг 3.5 – Функция сокрытия загружаемого модуля ядра

```

1 void hide(void)
2 {
3     if (flag_hidden)
4         return;
5
6     module_prev = THIS_MODULE->list.prev;
7     list_del(&THIS_MODULE->list);
8     flag_hidden = 1;
9
10    printk(">>> FIREWALL: module was hidden");
11 }

```

В листинге 3.6 приведена реализация функции возобновления видимости разработанного загружаемого модуля ядра ОС Linux.

Листинг 3.6 – Функция возобновления видимости загружаемого модуля ядра

```
1 void unhide(void)
2 {
3     if (!flag_hidden)
4         return;
5     list_add(&THIS_MODULE->list, module_prev);
6     flag_hidden = 0;
7     printk(">>> FIREWALL: module was exposed");
8 }
```

## 4 Исследовательский раздел

### 4.1 Команды

Для того, чтобы посмотреть все команды и формат задаваемых правил, необходимо вызвать **help**. На Рисунке 4.1 представлен результат.

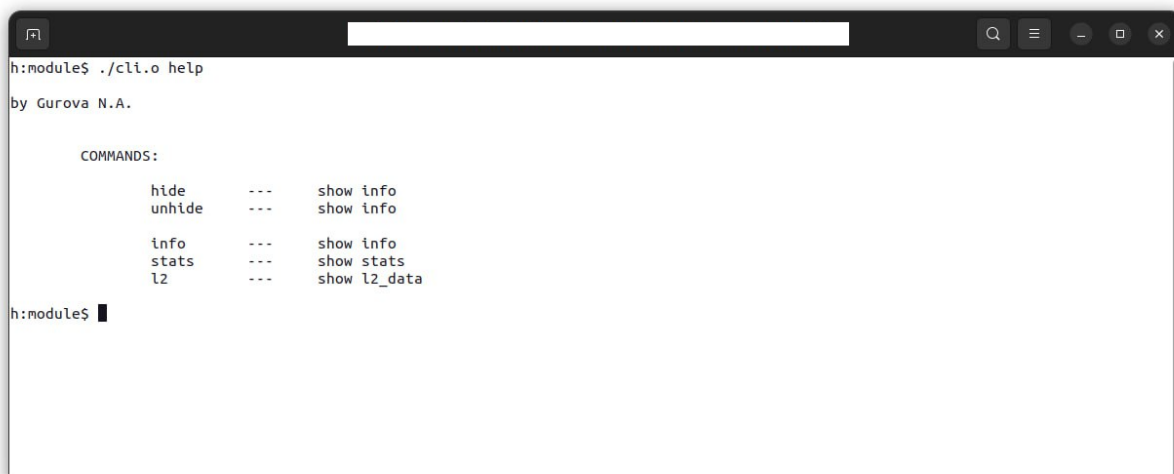


Рисунок 4.1 – Вызов команды help

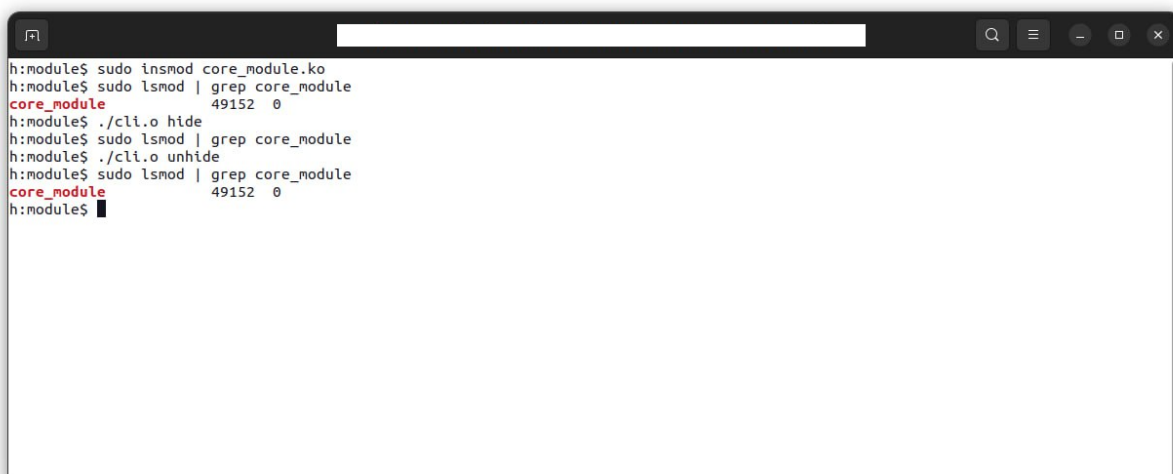
### 4.2 Видимость модуля

Для скрытия модуля следует вызвать команду **hide**, а для обратного действия **unhide**. На рисунке 4.2 демонстрируется следующее:

- модуль загружен и виден в системе;
- вызвана команда **hide**;
- модуль не отображается при вызове команды **lsmod**;
- вызвана команда **unhide**;
- модуль обнаруживается при вызове команды **lsmod**.

### 4.3 Вывод статистики по работе модуля

Для просмотра статистики по работе модуля предусмотрена команда **stats**. Эта команда позволяет просмотреть агрегированную статистику работы

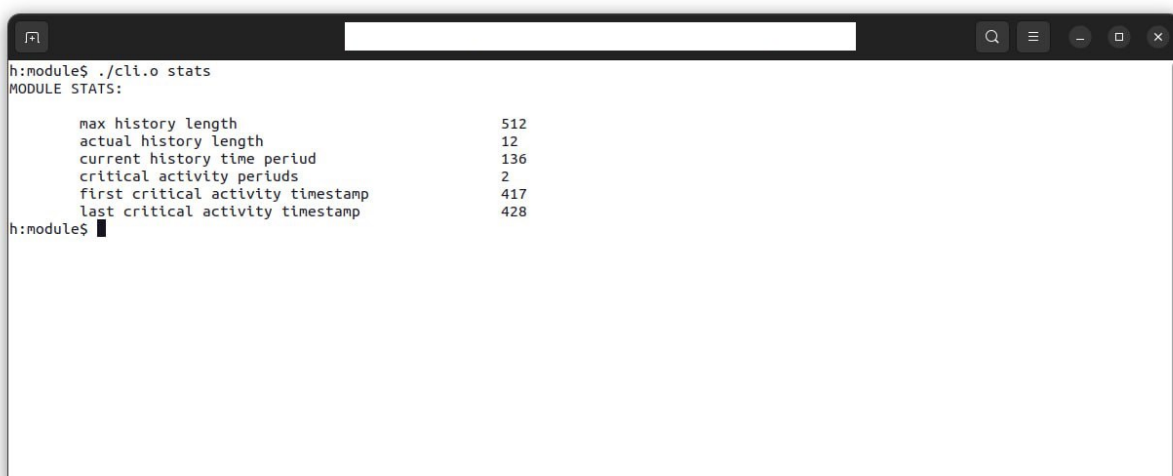


```
h:module$ sudo insmod core_module.ko
h:module$ sudo lsmod | grep core_module
core_module          49152  0
h:module$ ./cli.o hide
h:module$ sudo lsmod | grep core_module
h:module$ ./cli.o unhide
h:module$ sudo lsmod | grep core_module
core_module          49152  0
h:module$
```

Рисунок 4.2 – Управление видимостью модуля

модуля: максимальная длина исторических данных, текущая длина исторических данных, текущая длина исторических данных (в секундах), количество критических периодов, первая временная метка возникновения критически активного периода, последняя временная метка возникновения критически активного периода.

На рисунке 4.3 представлена демонстрация работы команды **stats**.



```
h:module$ ./cli.o stats
MODULE STATS:

max history length          512
actual history length       12
current history time period 136
critical activity periods    2
first critical activity timestamp 417
last critical activity timestamp 428
h:module$
```

Рисунок 4.3 – Вывод статистики по работе модуля

## 4.4 Просмотр исторических данных

Для просмотра собранных исторических данных предусмотрена команда **12**. Данная команда выводит собранные исторические данные в виде последовательности json-документов. Отчет о временном промежутке включает:

временную метку периода, количество уникальных источников трафика, общее количество перехваченных пакетов, общий размер перехваченного трафика (в байтах), среднее количество перехваченных пакетов, максимальное количество перехваченных пакетов, средний размер перехваченных пакетов (в байтах), максимальный размер перехваченных пакетов (в байтах), флаг наличия аномалия, название аномального параметра.

На рисунке 4.4 представлен результат выполнения команды **l2**.



```
h:module$ ./cli.o l2
[{"from_time": 1187, "unique_saddr_count": 4, "caught_packets_count": 21, "tot_len": 555268, "avg_catched_packets": 5, "max_catched_packets": 16, "avg_len": 138817, "max_len": 391426, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1201, "unique_saddr_count": 4, "caught_packets_count": 14, "tot_len": 485398, "avg_catched_packets": 3, "max_catched_packets": 7, "avg_len": 121349, "max_len": 240662, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1213, "unique_saddr_count": 4, "caught_packets_count": 16, "tot_len": 481287, "avg_catched_packets": 4, "max_catched_packets": 8, "avg_len": 120321, "max_len": 200966, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1226, "unique_saddr_count": 3, "caught_packets_count": 8, "tot_len": 327168, "avg_catched_packets": 2, "max_catched_packets": 4, "avg_len": 109056, "max_len": 112896, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1241, "unique_saddr_count": 4, "caught_packets_count": 13, "tot_len": 345369, "avg_catched_packets": 3, "max_catched_packets": 8, "avg_len": 86342, "max_len": 165888, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1251, "unique_saddr_count": 5, "caught_packets_count": 9, "tot_len": 280066, "avg_catched_packets": 1, "max_catched_packets": 3, "avg_len": 56013, "max_len": 93954, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1261, "unique_saddr_count": 1, "caught_packets_count": 2, "tot_len": 110592, "avg_catched_packets": 2, "max_catched_packets": 2, "avg_len": 110592, "max_len": 110592, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1272, "unique_saddr_count": 3, "caught_packets_count": 10, "tot_len": 330758, "avg_catched_packets": 3, "max_catched_packets": 6, "avg_len": 110252, "max_len": 165888, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1282, "unique_saddr_count": 6, "caught_packets_count": 10, "tot_len": 403968, "avg_catched_packets": 1, "max_catched_packets": 3, "avg_len": 67328, "max_len": 110592, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1296, "unique_saddr_count": 5, "caught_packets_count": 45, "tot_len": 812291, "avg_catched_packets": 9, "max_catched_packets": 33, "avg_len": 162458, "max_len": 487936, "anomaly": 1, "property": "tot_len", "type": "increase"}, {"from_time": 1306, "unique_saddr_count": 5, "caught_packets_count": 11, "tot_len": 399105, "avg_catched_packets": 2, "max_catched_packets": 5, "avg_len": 79821, "max_len": 174080, "anomaly": 0, "property": "undefined", "type": "undefined"}, {"from_time": 1316, "unique_saddr_count": 4, "caught_packets_count": 14, "tot_len": 466691, "avg_catched_packets": 3, "max_catched_packets": 14, "avg_len": 104943, "max_len": 200966, "anomaly": 0, "property": "undefined", "type": "undefined"}]
```

Рисунок 4.4 – Просмотр статистики

## 4.5 Вывод

В данном разделе была проведена демонстрация работы разработанного программного обеспечения.

В качестве результатов работы разработанного программного обеспечения приводятся изображения работы интерфейса командной строки.

Продемонстрированы: подсказка интерфейса командной строки, возможность изменения видимости загружаемого модуля ядра ОС Linux, возможность просмотра статистики работы загружаемого модуля для ОС Linux, возможность просмотра собранных исторических данных.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был определен способ перехвата входящих и исходящих пакетов — путем регистрации функций перехвата с использованием библиотеки Netfilter.

В качестве точек перехвата было решено использовать точку, которую проходят все исходящие пакеты (NF\_INET\_LOCAL\_IN).

Был разработан загружаемый модуль ядра ОС Linux, который осуществляет поиск атак во входящем сетевом трафике. Также был разработан интерфейс командной строки для управления разработанным загружаемым модулем для ядра ОС Linux. Предусмотрена возможность сокрытия модуля.

Продемонстрированы: подсказка интерфейса командной строки, возможность изменения видимости загружаемого модуля ядра ОС Linux, возможность просмотра статистики работы загружаемого модуля для ОС Linux, возможность просмотра собранных исторических данных.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Устройство TCP / Реализация SYN-flood атаки [Электронный ресурс]. — Режим доступа: <https://habr.com/ru/articles/782728/> (дата обращения: 19.01.2024).
2. Корнев Д. А., Лопин В. Н., Лузгин В. Г. Активные методы обнаружения SYN-flood атак // Ученые записки. Электронный научный журнал Курского государственного университета. — 2012. — 4 (24). — С. 64—70.
3. Рязанова Н. Ю., Курс лекций по дисциплине «Операционные системы» [Текст].
4. Документация NAPI. [Электронный ресурс]. — Режим доступа: <https://wiki.linuxfoundation.org/networking/napi> (дата обращения: 12.10.2024).
5. В. Мешков, Netfilter, журнал «Системный администратор». [Электронный ресурс]. — Режим доступа: <http://samag.ru/archive/article/169> (дата обращения: 12.10.2024).
6. The netfilter.org project. [Электронный ресурс]. — Режим доступа: <https://www.netfilter.org/> (дата обращения: 12.10.2022).
7. Карачанская Е. В., Соседова Н. И. Метод выявления аномалий сетевого трафика, основанный на его самоподобной структуре // Безопасность информационных технологий. — 2019. — Т. 26, № 1. — С. 98—110.
8. Qt Creator – A Cross-platform IDE for software development. [Электронный ресурс]. — Режим доступа: <https://www.qt.io/product/development-tools> (дата обращения: 13.10.2024).



## ПРИЛОЖЕНИЕ А

Листинг 4.1 – core\_module.h

```
1 #ifndef CORE_MODULE_H
2 #define CORE_MODULE_H
3
4
5 #define L2_DATA_DEV_PATH "/dev/course_work_l2_data"
6 #define STATS_DEV_PATH "/dev/course_work_stats"
7 #define CTRL_DEV_PATH "/dev/course_work_ctrl"
8
9 enum command_type {
10     UNDEFINED = 0,
11     HIDE = 1,
12     UNHIDE = 2,
13 };
14
15 struct command
16 {
17     enum command_type type;
18 };
19
20 enum l2_property {
21     L2_PROPERTY_UNDEFINED = 0,
22     L2_PROPERTY_UNIQUE_SADDRS_COUNT = 1,
23     L2_PROPERTY_CATCHED_PACKETS_COUNT = 2,
24     L2_PROPERTY_TOT_LEN = 3,
25     L2_PROPERTY_AVG_CATCHED_PACKETS = 4,
26     L2_PROPERTY_MAX_CATCHED_PACKETS = 5,
27     L2_PROPERTY_AVG_LEN = 6,
28     L2_PROPERTY_MAX_LEN = 7,
29     L2_PROPERTY_TOT_TCP = 8,
30     L2_PROPERTY_TOT_SYN = 9,
31 };
32 enum l2_crit_behaviour_type {
33     L2_CRIT_BEHAVIOUR_TYPE_UNDEFINED = 0,
34     L2_CRIT_BEHAVIOUR_TYPE_INCREASE = 1,
35     L2_CRIT_BEHAVIOUR_TYPE_FALL = 2,
36 };
37
38 struct l2_data_slice
```

```

39 {
40     uint64_t from_time;
41     uint32_t unique_saddr_count;
42     uint32_t caught_packets_count;
43     uint32_t tot_len;
44     uint32_t avg_caught_packets;
45     uint32_t max_caught_packets;
46     uint32_t avg_len;
47     uint32_t max_len;
48     int crit_behaviour;
49     enum l2_property property;
50     enum l2_crit_behaviour_type type;
51     uint32_t tot_tcp;
52     uint32_t tot_syn;
53 };
54
55 struct module_stats
56 {
57     uint32_t history_length;
58     uint32_t current_history_length;
59     uint64_t current_period_ns;
60     uint32_t crit_behaviour_count;
61     uint64_t first_crit_behaviour_ns;
62     uint64_t last_crit_behaviour_ns;
63 };
64
65 #endif //CORE_MODULE_H

```

## ПРИЛОЖЕНИЕ Б

Листинг 4.2 – core\_module.c

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <linux/fs.h>
4 #include <linux/init.h>
5 #include <linux/list.h>
6 #include <linux/slab.h>
7 #include <linux/cdev.h>
8 #include <linux/device.h>
9 #include <linux/types.h>
10
11 #include <linux/netfilter_ipv4.h>
12 #include <linux/netfilter.h>
13 #include <linux/in.h>
14 #include <linux/ip.h>
15 #include <linux/tcp.h>
16 #include <linux/udp.h>
17
18 #include <linux/fcntl.h>
19 #include <linux/delay.h>
20 #include <linux/syscalls.h>
21
22 #include <linux/miscdevice.h>
23 #include <linux/stat.h>
24
25 #include <linux/string.h>
26 #include <linux/timekeeping.h>
27
28
29 #include "core_module.h"
30
31
32 #define MODULE_DMESG_PREFIX ">>>—————> "
33 #define DEVICE_L2_DATA_FNAME "cource_work_l2_data"
34 #define DEVICE_STATS_FNAME "cource_work_stats"
35 #define DEVICE_CTRL_FNAME "cource_work_ctrl"
36
37
38 #define IP_POS(ip, pos) (ip >> ((8 * (3 - pos))) & 0xFF)
```

```

39 #define SAME_ADDR(ip1, ip2) ((ip1 ^ ip2) == 0)
40
41 MODULE_LICENSE("GPL");
42 MODULE_AUTHOR("Gurova N.A.");
43
44 uint64_t dump_ns_periud = 10000000000;
45
46 #define FIRST_LAYER_TABLE_LENGTH 64
47 struct first_layer_schema {
48     uint32_t unique_saddr;
49     uint32_t caught_packets_count;
50     uint32_t tot_len;
51     uint32_t caught_tcp;
52     uint32_t caught_syn;
53 };
54 struct first_layer_schema
55     first_layer_table[FIRST_LAYER_TABLE_LENGTH];
56 size_t first_layer_table_next_index = 0;
57 uint64_t from_time = 0; // ns
58
59 #define SECOND_LAYER_TABLE_LENGTH 512
60 enum crit_behaviour_property {
61     CRIT_BEHAVIOUR_PROPERTY_UNDEFINED = 0,
62     CRIT_BEHAVIOUR_PROPERTY_UNIQUE_SADDRS_COUNT = 1,
63     CRIT_BEHAVIOUR_PROPERTY_CATCHED_PACKETS_COUNT = 2,
64     CRIT_BEHAVIOUR_PROPERTY_TOT_LEN = 3,
65     CRIT_BEHAVIOUR_PROPERTY_AVG_CATCHED_PACKETS = 4,
66     CRIT_BEHAVIOUR_PROPERTY_MAX_CATCHED_PACKETS = 5,
67     CRIT_BEHAVIOUR_PROPERTY_AVG_LEN = 6,
68     CRIT_BEHAVIOUR_PROPERTY_MAX_LEN = 7,
69     CRIT_BEHAVIOUR_PROPERTY_TOT_TCP = 8,
70     CRIT_BEHAVIOUR_PROPERTY_TOT_SYN = 9,
71 };
72 enum crit_behaviour_type {
73     CRIT_BEHAVIOUR_TYPE_UNDEFINED = 0,
74     CRIT_BEHAVIOUR_TYPE_INCREASE = 1,
75     CRIT_BEHAVIOUR_TYPE_FALL = 2,
76 };
77 struct second_layer_schema {
78     uint64_t from_time;
79     uint32_t unique_saddr_count;

```

```

79     uint32_t  caught_packets_count;
80     uint32_t  tot_len;
81     uint32_t  avg_caatched_packets;
82     uint32_t  max_caatched_packets;
83     uint32_t  avg_len;
84     uint32_t  max_len;
85     int crit_behaviour;
86     enum crit_behaviour_property property;
87     enum crit_behaviour_type type;
88     uint32_t  tot_tcp;
89     uint32_t  tot_syn;
90 };
91 struct second_layer_schema
92     second_layer_table[SECOND_LAYER_TABLE_LENGTH];
93 size_t second_layer_table_next_index = 0;
94 size_t second_layer_table_read_index = 0;
95
96 struct second_layer_schema initial_vec = {
97     .unique_saddr_count = 10,
98     .caught_packets_count = 100,
99     .tot_len = 1000000,
100    .avg_caatched_packets = 50,
101    .max_caatched_packets = 50,
102    .avg_len = 300000,
103    .max_len = 300000,
104    .tot_tcp = 100,
105    .tot_syn = 100,
106 };
107
108 int eps_percent = 30;
109
110 void inspect_last(void) {
111     if (second_layer_table_next_index < 2) {
112         return;
113     }
114
115     struct second_layer_schema avg_values = {
116         .unique_saddr_count = 0,
117         .caught_packets_count = 0,
118         .tot_len = 0,
119         .avg_caatched_packets = 0,

```

```

119         .max_catched_packets = 0,
120         .avg_len = 0,
121         .max_len = 0,
122         .tot_tcp = 0,
123         .tot_syn = 0,
124     };
125
126     for (int i = 0; i < second_layer_table_next_index - 1; i++) {
127         avg_values.unique_saddr_count +=
128             second_layer_table[i].unique_saddr_count;
129         avg_values.catched_packets_count +=
130             second_layer_table[i].catched_packets_count;
131         avg_values.tot_len += second_layer_table[i].tot_len;
132         avg_values.avg_catched_packets +=
133             second_layer_table[i].avg_catched_packets;
134         avg_values.max_catched_packets +=
135             second_layer_table[i].max_catched_packets;
136         avg_values.avg_len += second_layer_table[i].avg_len;
137         avg_values.max_len += second_layer_table[i].max_len;
138         avg_values.tot_tcp += second_layer_table[i].tot_tcp;
139         avg_values.tot_syn += second_layer_table[i].tot_syn;
140     }
141
142     avg_values.unique_saddr_count /= (second_layer_table_next_index
143         - 1);
144     avg_values.catched_packets_count /=
145         (second_layer_table_next_index - 1);
146     avg_values.tot_len /= (second_layer_table_next_index - 1);
147     avg_values.avg_catched_packets /=
148         (second_layer_table_next_index - 1);
149     avg_values.max_catched_packets /=
150         (second_layer_table_next_index - 1);
151     avg_values.avg_len /= (second_layer_table_next_index - 1);
152     avg_values.max_len /= (second_layer_table_next_index - 1);
153     avg_values.tot_tcp /= (second_layer_table_next_index - 1);
154     avg_values.tot_syn /= (second_layer_table_next_index - 1);
155
156     struct second_layer_schema last =
157         second_layer_table[second_layer_table_next_index - 1];
158
159     uint64_t limit;

```

```

151
152     limit = avg_values.unique_saddr_count * (100 + eps_percent) /
153           100 + initial_vec.unique_saddr_count;
154     if (last.unique_saddr_count > limit) {
155         second_layer_table[second_layer_table_next_index -
156                             1].crit_behaviour = 1;
157         second_layer_table[second_layer_table_next_index -
158                             1].property =
159             CRIT_BEHAVIOUR_PROPERTY_UNIQUE_SADDRS_COUNT;
160         second_layer_table[second_layer_table_next_index - 1].type
161             = CRIT_BEHAVIOUR_TYPE_INCREASE;
162         return;
163     }
164
165     limit = avg_values.catched_packets_count * (100 + eps_percent)
166           / 100 + initial_vec.catched_packets_count;
167     if (last.catched_packets_count > limit) {
168         second_layer_table[second_layer_table_next_index -
169                             1].crit_behaviour = 1;
170         second_layer_table[second_layer_table_next_index -
171                             1].property =
172             CRIT_BEHAVIOUR_PROPERTY_CATCHED_PACKETS_COUNT;
173         second_layer_table[second_layer_table_next_index - 1].type
174             = CRIT_BEHAVIOUR_TYPE_INCREASE;
175         return;
176     }
177
178     limit = avg_values.tot_len * (100 + eps_percent) / 100 +
179           initial_vec.tot_len;
180     if (last.tot_len > limit) {
181         second_layer_table[second_layer_table_next_index -
182                             1].crit_behaviour = 1;
183         second_layer_table[second_layer_table_next_index -
184                             1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_LEN;
185         second_layer_table[second_layer_table_next_index - 1].type
186             = CRIT_BEHAVIOUR_TYPE_INCREASE;
187         return;
188     }
189
190     limit = avg_values.avg_catched_packets * (100 + eps_percent) /
191           100 + initial_vec.avg_catched_packets;

```

```

177     if (last.avg_catched_packets > limit) {
178         second_layer_table[second_layer_table_next_index -
179             1].crit_behaviour = 1;
180         second_layer_table[second_layer_table_next_index -
181             1].property =
182             CRIT_BEHAVIOUR_PROPERTY_AVG_CATCHED_PACKETS;
183         second_layer_table[second_layer_table_next_index - 1].type
184             = CRIT_BEHAVIOUR_TYPE_INCREASE;
185         return;
186     }
187
188     limit = avg_values.max_catched_packets * (100 + eps_percent) /
189         100 + initial_vec.max_catched_packets;
190     if (last.max_catched_packets > limit) {
191         second_layer_table[second_layer_table_next_index -
192             1].crit_behaviour = 1;
193         second_layer_table[second_layer_table_next_index -
194             1].property =
195             CRIT_BEHAVIOUR_PROPERTY_MAX_CATCHED_PACKETS;
196         second_layer_table[second_layer_table_next_index - 1].type
197             = CRIT_BEHAVIOUR_TYPE_INCREASE;
198         return;
199     }
200
201     limit = avg_values.avg_len * (100 + eps_percent) / 100 +
202         initial_vec.avg_len;
203     if (last.avg_len > limit) {
204         second_layer_table[second_layer_table_next_index -
205             1].crit_behaviour = 1;
206         second_layer_table[second_layer_table_next_index -
207             1].property = CRIT_BEHAVIOUR_PROPERTY_AVG_LEN;
208         second_layer_table[second_layer_table_next_index - 1].type
209             = CRIT_BEHAVIOUR_TYPE_INCREASE;
210         return;
211     }
212
213     limit = avg_values.max_len * (100 + eps_percent) / 100 +
214         initial_vec.max_len;
215     if (last.max_len > limit) {
216         second_layer_table[second_layer_table_next_index -
217             1].crit_behaviour = 1;

```



```

203     second_layer_table[second_layer_table_next_index -
        1].property = CRIT_BEHAVIOUR_PROPERTY_MAX_LEN;
204     second_layer_table[second_layer_table_next_index - 1].type
        = CRIT_BEHAVIOUR_TYPE_INCREASE;
205     return;
206 }
207
208     limit = avg_values.tot_tcp * (100 + eps_percent) / 100 +
        initial_vec.tot_tcp;
209     if (last.tot_tcp > limit) {
210         second_layer_table[second_layer_table_next_index -
            1].crit_behaviour = 1;
211         second_layer_table[second_layer_table_next_index -
            1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_TCP;
212         second_layer_table[second_layer_table_next_index - 1].type
            = CRIT_BEHAVIOUR_TYPE_INCREASE;
213         return;
214     }
215
216     limit = avg_values.tot_syn * (100 + eps_percent) / 100 +
        initial_vec.tot_syn;
217     if (last.tot_syn > limit) {
218         second_layer_table[second_layer_table_next_index -
            1].crit_behaviour = 1;
219         second_layer_table[second_layer_table_next_index -
            1].property = CRIT_BEHAVIOUR_PROPERTY_TOT_SYN;
220         second_layer_table[second_layer_table_next_index - 1].type
            = CRIT_BEHAVIOUR_TYPE_INCREASE;
221         return;
222     }
223 }
224
225 void dump_first_layer_table(void) {
226     if (second_layer_table_next_index >= SECOND_LAYER_TABLE_LENGTH)
227     {
228         for (int i = 0; i < SECOND_LAYER_TABLE_LENGTH - 1; i++) {
229             second_layer_table[i] = second_layer_table[i + 1];
230         }
231         second_layer_table_next_index -= 1;
232     }

```

```

233
234     if (second_layer_table_next_index < SECOND_LAYER_TABLE_LENGTH) {
235         uint32_t caught_packets_count = 0;
236         uint32_t tot_len = 0;
237         uint32_t avg_caught_packets = 0;
238         uint32_t max_caught_packets = 0;
239         uint32_t avg_len = 0;
240         uint32_t max_len = 0;
241         uint32_t tot_tcp = 0;
242         uint32_t tot_syn = 0;
243
244         for (int i = 0; i < first_layer_table_next_index; i++) {
245             caught_packets_count +=
246                 first_layer_table[i].caught_packets_count;
247             tot_len += first_layer_table[i].tot_len;
248             if (max_caught_packets <
249                 first_layer_table[i].caught_packets_count) {
250                 max_caught_packets =
251                     first_layer_table[i].caught_packets_count; }
252             if (max_len < first_layer_table[i].tot_len) { max_len =
253                 first_layer_table[i].tot_len; }
254
255             tot_tcp += first_layer_table[i].caught_tcp;
256             tot_syn += first_layer_table[i].caught_syn;
257         }
258
259         avg_caught_packets = caught_packets_count /
260             first_layer_table_next_index;
261         avg_len = tot_len / first_layer_table_next_index;
262
263         second_layer_table[second_layer_table_next_index].from_time
264             = ktime_get_seconds();
265         second_layer_table[second_layer_table_next_index]
266             .unique_saddr_count = first_layer_table_next_index;
267         second_layer_table[second_layer_table_next_index]
268             .caught_packets_count = caught_packets_count;
269         second_layer_table[second_layer_table_next_index]
270             .tot_len = tot_len;
271         second_layer_table[second_layer_table_next_index]
272             .avg_caught_packets = avg_caught_packets;
273         second_layer_table[second_layer_table_next_index]

```

```

267         .max_catched_packets = max_catched_packets;
268     second_layer_table[second_layer_table_next_index]
269         .avg_len = avg_len;
270     second_layer_table[second_layer_table_next_index]
271         .max_len = max_len;
272
273     second_layer_table[second_layer_table_next_index]
274         .crit_behaviour = 0;
275     second_layer_table[second_layer_table_next_index]
276         .property = CRIT_BEHAVIOUR_PROPERTY_UNDEFINED;
277     second_layer_table[second_layer_table_next_index]
278         .type = CRIT_BEHAVIOUR_TYPE_UNDEFINED;
279
280     second_layer_table[second_layer_table_next_index]
281         .tot_tcp = tot_tcp;
282     second_layer_table[second_layer_table_next_index]
283         .tot_syn = tot_syn;
284
285     first_layer_table_next_index = 0;
286     second_layer_table_next_index += 1;
287
288     inspect_last();
289 }
290 }
291
292 void add_packet_info(uint32_t saddr, uint32_t tot_len, uint32_t
    is_tcp, uint32_t is_syn) {
293
294     if (ktime_get_ns() - from_time > dump_ns_periud) {
295         if (from_time != 0) dump_first_layer_table();
296
297         from_time = ktime_get_ns();
298     }
299
300     int records_found = 0;
301
302     for (int i = 0; i < first_layer_table_next_index; i++) {
303         if (first_layer_table[i].unique_saddr == saddr) {
304             records_found = 1;
305
306             first_layer_table[i].catched_packets_count += 1;

```

```

307         first_layer_table[i].tot_len += tot_len;
308
309         if (is_tcp) first_layer_table[i].caught_tcp += 1;
310         if (is_syn) first_layer_table[i].caught_syn += 1;
311
312         break;
313     }
314 }
315
316 if (records_found != 1 && first_layer_table_next_index <
    FIRST_LAYER_TABLE_LENGTH) {
317     first_layer_table[first_layer_table_next_index]
318         .unique_saddr = saddr;
319     first_layer_table[first_layer_table_next_index]
320         .caught_packets_count = 1;
321     first_layer_table[first_layer_table_next_index]
322         .tot_len = tot_len;
323
324     if (is_tcp) {
325         first_layer_table[first_layer_table_next_index]
326             .caught_tcp = 1;
327     } else {
328         first_layer_table[first_layer_table_next_index]
329             .caught_tcp = 0;
330     }
331
332     if (is_syn) {
333         first_layer_table[first_layer_table_next_index]
334             .caught_syn = 1;
335     } else {
336         first_layer_table[first_layer_table_next_index]
337             .caught_syn = 0;
338     }
339
340     first_layer_table_next_index += 1;
341 }
342 }
343
344
345 static unsigned int catch_traffic(void *priv, struct sk_buff *skb,
    const struct nf_hook_state *state)

```

```

346 {
347     struct iphdr *iph;  /* An IPv4 packet header */
348     struct tcphdr *tcph; /* An TCP packet header */
349     uint32_t saddr;
350     uint32_t tot_len;
351     uint32_t is_tcp = 0;
352     uint32_t is_syn = 0;
353
354     if (!skb) return NF_ACCEPT;
355
356     iph = (struct iphdr *)skb_network_header(skb);
357     if (iph == NULL) return NF_ACCEPT;
358
359     saddr = iph->saddr;
360     tot_len = iph->tot_len;
361
362     if (iph->protocol == IPPROTO_TCP) {
363         tcph = (struct tcphdr *) (skb_transport_header(skb));
364         is_tcp = 1;
365         if (tcph->syn) is_syn = 1;
366     }
367
368     add_packet_info(saddr, tot_len, is_tcp, is_syn);
369
370     return NF_ACCEPT;  /* the packet passes, continue iterations */
371 }
372
373 static struct nf_hook_ops module_hook_ops =
374 {
375     .hook = catch_traffic,
376     .pf = PF_INET,
377     .hooknum = NF_INET_LOCAL_IN,
378     .priority = NF_IP_PRI_FIRST
379 };
380
381 // hide && unhide
382
383 struct list_head *module_prev;
384 int flag_hidden = 0;
385
386 void hide(void) {

```

```

387     if (flag_hidden)
388         return;
389
390     module_prev = THIS_MODULE->list.prev;
391     list_del(&THIS_MODULE->list);
392     flag_hidden = 1;
393
394     printk("module was hidden");
395 }
396
397 void unhide(void) {
398     if (!flag_hidden)
399         return;
400
401     list_add(&THIS_MODULE->list, module_prev);
402     flag_hidden = 0;
403
404     printk("module was exposed");
405 }
406
407 // misc devices description
408
409 ssize_t write_stub(struct file *filp, const char __user *buff,
410     size_t count, loff_t *f_pos) { return 0; }
411 ssize_t read_stub(struct file *filp, char __user *buff, size_t
412     count, loff_t *f_pos) { return 0; }
413 int open_stub(struct inode *inode, struct file *file) { return 0; }
414 int release_stub(struct inode *inode, struct file *file) { return
415     0; }
416
417 ssize_t read_l2_data(struct file *filp, char __user *buff, size_t
418     count, loff_t *f_pos) {
419     if (count != sizeof(struct l2_data_slice)) {
420         printk(MODULE_DMESG_PREFIX "[ERROR] invalid count");
421         return 0;
422     }
423
424     if (second_layer_table_read_index >=
425         second_layer_table_next_index) {
426         second_layer_table_read_index = 0;
427         return 0;

```

```

423     }
424
425     struct l2_data_slice l2_ds = {
426         .from_time =
427             second_layer_table[second_layer_table_read_index]
428                 .from_time ,
429         .unique_saddr_count =
430             second_layer_table[second_layer_table_read_index]
431                 .unique_saddr_count ,
432         .caught_packets_count =
433             second_layer_table[second_layer_table_read_index]
434                 .caught_packets_count ,
435         .tot_len = second_layer_table[second_layer_table_read_index]
436                 .tot_len ,
437         .avg_caught_packets =
438             second_layer_table[second_layer_table_read_index]
439                 .avg_caught_packets ,
440         .max_caught_packets =
441             second_layer_table[second_layer_table_read_index]
442                 .max_caught_packets ,
443         .avg_len = second_layer_table[second_layer_table_read_index]
444                 .avg_len ,
445         .max_len = second_layer_table[second_layer_table_read_index]
446                 .max_len ,
447         .crit_behaviour =
448             second_layer_table[second_layer_table_read_index]
449                 .crit_behaviour ,
450         .property =
451             second_layer_table[second_layer_table_read_index]
452                 .property ,
453         .type = second_layer_table[second_layer_table_read_index]
454                 .type ,
455         .tot_tcp = second_layer_table[second_layer_table_read_index]
456                 .tot_tcp ,
457         .tot_syn = second_layer_table[second_layer_table_read_index]
458                 .tot_syn ,
459     };
460
461     if (copy_to_user(buff, (char *) &l2_ds, sizeof(struct
462         l2_data_slice)))
463     {

```

```

456         printk(MODULE_DMESG_PREFIX "[ERROR] copy_to_user error");
457         return 0;
458     }
459
460     second_layer_table_read_index += 1;
461
462     return sizeof(struct l2_data_slice);
463 }
464
465 ssize_t read_stats(struct file *filp, char __user *buff, size_t
count, loff_t *f_pos) {
466     if (count != sizeof(struct module_stats)) {
467         printk(MODULE_DMESG_PREFIX "[ERROR] invalid count");
468         return 0;
469     }
470
471     uint32_t history_length = SECOND_LAYER_TABLE_LENGTH;
472     uint32_t current_history_length = second_layer_table_next_index;
473     uint64_t current_periud_ns = 0;
474     uint32_t crit_behaviour_count = 0;
475     uint64_t first_crit_behaviour_ns = 0;
476     uint64_t last_crit_behaviour_ns = 0;
477
478     for (int i = 0; i < second_layer_table_next_index; i++) {
479         if (second_layer_table[i].crit_behaviour > 0) {
480             crit_behaviour_count += 1;
481             if (first_crit_behaviour_ns == 0) {
482                 first_crit_behaviour_ns =
                    second_layer_table[i].from_time;
483             }
484             last_crit_behaviour_ns =
                    second_layer_table[i].from_time;
485         }
486     }
487
488     uint64_t per_start = 0, per_finish = 0;
489     if (second_layer_table_next_index > 0) {
490         per_start = second_layer_table[0].from_time;
491         per_finish =
            second_layer_table[second_layer_table_next_index -
                1].from_time;

```



```

492         // printk(MODULE_DMESG_PREFIX "%lld %lld", per_start,
           per_finish);
493     }
494     current_period_ns = per_finish - per_start;
495
496     struct module_stats stats = {
497         .history_length = history_length,
498         .current_history_length = current_history_length,
499         .current_period_ns = current_period_ns,
500         .crit_behaviour_count = crit_behaviour_count,
501         .first_crit_behaviour_ns = first_crit_behaviour_ns,
502         .last_crit_behaviour_ns = last_crit_behaviour_ns,
503     };
504
505     if (copy_to_user(buff, (char *) &stats, sizeof(struct
        module_stats)))
506     {
507         printk(MODULE_DMESG_PREFIX "[ERROR] copy_to_user error");
508         return 0;
509     }
510
511     return sizeof(struct module_stats);
512 }
513
514 ssize_t write_command(struct file *filp, const char __user *buff,
        size_t count, loff_t *f_pos)
515 {
516     struct command cmd;
517
518     if (count != sizeof(struct command))
519     {
520         printk(MODULE_DMESG_PREFIX "[ERROR] incorrect command");
521         return 0;
522     }
523
524     if (copy_from_user(&cmd, buff, sizeof(struct command)))
525     {
526         printk(MODULE_DMESG_PREFIX "[ERROR] copy_from_user error");
527         return 0;
528     }
529

```

```

530     if (cmd.type == HIDE) {
531         hide();
532     }
533
534     if (cmd.type == UNHIDE) {
535         unhide();
536     }
537
538     return 0;
539 }
540
541 static struct file_operations l2_data_fops = {
542     .owner = THIS_MODULE,
543     .read = read_l2_data,
544     .write = write_stub,
545     .open = open_stub,
546     .release = release_stub,
547 };
548
549 static struct file_operations stats_fops = {
550     .owner = THIS_MODULE,
551     .read = read_stats,
552     .write = write_stub,
553     .open = open_stub,
554     .release = release_stub,
555 };
556
557 static struct file_operations ctrl_fops = {
558     .owner = THIS_MODULE,
559     .read = read_stub,
560     .write = write_command,
561     .open = open_stub,
562     .release = release_stub,
563 };
564
565 struct miscdevice dev_l2_data = {
566     .minor = MISC_DYNAMIC_MINOR,
567     .name = DEVICE_L2_DATA_FNAME,
568     .fops = &l2_data_fops,
569     .mode = S_IRWXU | S_IWGRP | S_IWOTH | S_IROTH,
570 };

```

```

571
572 struct miscdevice dev_stats = {
573     .minor = MISC_DYNAMIC_MINOR,
574     .name = DEVICE_STATS_FNAME,
575     .fops = &stats_fops ,
576     .mode = S_IRWXU | S_IWGRP | S_IWOTH | S_IROTH,
577 };
578
579 struct miscdevice dev_ctrl = {
580     .minor = MISC_DYNAMIC_MINOR,
581     .name = DEVICE_CTRL_FNAME,
582     .fops = &ctrl_fops ,
583     .mode = S_IRWXU | S_IWGRP | S_IWOTH | S_IROTH,
584 };
585
586 static int __init my_module_init(void) {
587     int rc = 0;
588
589     nf_register_net_hook(&init_net , &module_hook_ops);
590
591     rc = misc_register(&dev_l2_data);
592     if (rc)
593     {
594         printk(MODULE_DMESG_PREFIX "[ERROR] registration was
595             failed");
596         return rc;
597     }
598
599     rc = misc_register(&dev_stats);
600     if (rc)
601     {
602         printk(MODULE_DMESG_PREFIX "[ERROR] registration was
603             failed");
604         return rc;
605     }
606
607     rc = misc_register(&dev_ctrl);
608     if (rc)
609     {
610         printk(MODULE_DMESG_PREFIX "[ERROR] registration was
611             failed");

```

```

609         return rc;
610     }
611
612     printk(MODULE_DMESG_PREFIX "module was loaded");
613
614     return 0;
615 }
616
617 static void __exit my_module_exit(void) {
618
619     nf_unregister_net_hook(&init_net, &module_hook_ops);
620
621     misc_deregister(&dev_l2_data);
622     misc_deregister(&dev_stats);
623     misc_deregister(&dev_ctrl);
624
625     printk(MODULE_DMESG_PREFIX "module was unloaded\n");
626 }
627
628 module_init(my_module_init);
629 module_exit(my_module_exit);

```

## ПРИЛОЖЕНИЕ В

Листинг 4.3 – cli.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <getopt.h>
4 #include <arpa/inet.h>
5 #include <netdb.h>
6 #include <limits.h>
7 #include <string.h>
8 #include <fcntl.h>
9
10 #include <unistd.h>
11
12 #include "core_module.h"
13
14 #define PROPERTY_COUNT 10
15 enum l2_property properties[] = {L2_PROPERTY_UNDEFINED,
    L2_PROPERTY_UNIQUE_SADDRS_COUNT,
    L2_PROPERTY_CATCHED_PACKETS_COUNT, L2_PROPERTY_TOT_LEN,
16    L2_PROPERTY_AVG_CATCHED_PACKETS,
    L2_PROPERTY_MAX_CATCHED_PACKETS, L2_PROPERTY_AVG_LEN,
    L2_PROPERTY_MAX_LEN, L2_PROPERTY_TOT_TCP,
    L2_PROPERTY_TOT_SYN};
17 char *property_names[] = {"undefined", "saddr_count",
    "caught_packet_count", "tot_len", "avg_caught_packets",
18    "max_caught_packets", "avg_len", "max_len", "tot_tcp", "tot_syn"};
19 char *get_property_name(enum l2_property prop) {
20     for (int i = 0; i < PROPERTY_COUNT; i++) {
21         if (properties[i] == prop) {
22             return property_names[i];
23         }
24     }
25     return property_names[0];
26 }
27
28 #define crit_behaviour_TYPE_COUNT 3
29 enum l2_crit_behaviour_type crit_behaviour_types[] =
    {L2_CRIT_BEHAVIOUR_TYPE_UNDEFINED,
    L2_CRIT_BEHAVIOUR_TYPE_INCREASE, L2_CRIT_BEHAVIOUR_TYPE_FALL};
```

```

30 char *crit_behaviour_type_names[] = {"undefined", "increase",
    "fall"};
31 char *get_crit_behaviour_type_name(enum l2_crit_behaviour_type at) {
32     for (int i = 0; i < crit_behaviour_TYPE_COUNT; i++) {
33         if (crit_behaviour_types[i] == at) {
34             return crit_behaviour_type_names[i];
35         }
36     }
37     return crit_behaviour_type_names[0];
38 }
39
40 void print_info(void) {
41     printf(
42         "\nby Gurova N.A.\n\n"
43         "\tCOMMANDS:\n\n"
44         "\t\tthide      —      hide\n"
45         "\t\ttunhide     —      unhide\n\n"
46         "\t\ttinfo       —      show info\n"
47         "\t\ttstats      —      show stats\n"
48         "\t\ttl2         —      show l2_data\n\n"
49     );
50 }
51
52 void print_l2_data_slice(struct l2_data_slice *l2_ds) {
53     printf("{
54     \"from_time\": %d,
55     \"unique_saddr_count\": %d,
56     \"caught_packets_count\": %d,
57     \"tot_len\": %d,
58     \"avg_caught_packets\": %d,
59     \"max_caught_packets\": %d,
60     \"avg_len\": %d,
61     \"max_len\": %d,
62     \"crit\": %d,
63     \"property\": \"%s\",
64     \"crit_type\": \"%s\",
65     \"tot_tcp\": \"%d\",
66     \"tot_syn\": \"%d\"
67     }\n", l2_ds->from_time, l2_ds->unique_saddr_count,
        l2_ds->caught_packets_count,

```

```

68     l2_ds->tot_len, l2_ds->avg_catched_packets,
        l2_ds->max_catched_packets, l2_ds->avg_len,
69     l2_ds->max_len, l2_ds->crit_behaviour,
        get_property_name(l2_ds->property),
        get_crit_behaviour_type_name(l2_ds->type),
70     l2_ds->tot_tcp, l2_ds->tot_syn);
71 }
72
73 int show_l2_data()
74 {
75     struct l2_data_slice ds;
76     int rb;
77
78     int fd = open(L2_DATA_DEV_PATH, O_RDONLY);
79     if (fd < 0) return 1;
80
81     size_t l2_ds_size = sizeof(struct l2_data_slice);
82
83     while ((rb = read(fd, &ds, l2_ds_size)) > 0) {
84         if (rb != l2_ds_size) break;
85
86         print_l2_data_slice(&ds);
87     }
88
89     close(fd);
90
91     return 0;
92 }
93
94 int show_stats()
95 {
96     struct module_stats stats;
97     int rb;
98
99     int fd = open(STATS_DEV_PATH, O_RDONLY);
100    if (fd < 0) return 1;
101
102    size_t stats_size = sizeof(struct module_stats);
103
104    if (read(fd, &stats, stats_size) != stats_size) {
105        close(fd);

```

```

106         return 1;
107     }
108
109     close(fd);
110
111     printf(
112         "MODULE STATS:\n\n"
113         "\tmax history length\t\t\t\t\t%d\n"
114         "\tactual history length\t\t\t\t\t%d\n"
115         "\tcurrent history time period\t\t\t\t\t%d\n"
116         "\tcritical activity periods\t\t\t\t\t%d\n"
117         "\tfirst critical activity timestamp\t\t\t\t\t%d\n"
118         "\tlast critical activity timestamp\t\t\t\t\t%d\n",
119         stats.history_length, stats.current_history_length,
120         stats.current_period_ns,
121         stats.crit_behaviour_count, stats.first_crit_behaviour_ns,
122         stats.last_crit_behaviour_ns
123     );
124
125     return 0;
126 }
127
128 void hide_unhide(int make_hidden) {
129     struct command cmd;
130     if (make_hidden == 1) {
131         cmd.type = HIDE;
132     } else {
133         cmd.type = UNHIDE;
134     }
135
136     int fd = open(CTRL_DEV_PATH, O_WRONLY | O_APPEND);
137     if (fd < 0) return;
138
139     write(fd, &cmd, sizeof(struct command));
140
141     close(fd);
142 }
143
144 int main(int argc, char *argv[])
145 {
146     if (argc != 2) {

```



```
145     print_info();
146     return 1;
147 }
148
149 if (!strcmp(argv[1], "info")) {
150     print_info();
151     return 0;
152 }
153
154 if (!strcmp(argv[1], "stats")) {
155     return show_stats();
156 }
157
158 if (!strcmp(argv[1], "l2")) {
159     return show_l2_data();
160 }
161
162 if (!strcmp(argv[1], "hide")) {
163     hide_unhide(1);
164     return 0;
165 }
166
167 if (!strcmp(argv[1], "unhide")) {
168     hide_unhide(0);
169     return 0;
170 }
171
172 print_info();
173 return 1;
174 }
```

## ПРИЛОЖЕНИЕ Г

Листинг 4.4 – Makefile

```
1 obj-m += core_module.o
2
3 all: cli.o core_module.o
4
5 cli.o: cli.c core_module.h
6     gcc -o cli.o cli.c
7
8 core_module.o: core_module.c
9     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
10
11 clean:
12     rm -rf cli *.o
13     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```