

Федеральное государственное автономное образовательное  
учреждение высшего образования



**ПОЛИТЕХ**  
Санкт-Петербургский  
политехнический университет  
Петра Великого

Институт компьютерных наук и технологий  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторному практикуму**  
по дисциплине "Сетевая безопасность"

Выполнил студент гр. 3540901/21501

\_\_\_\_\_ С.А.Мартынов  
(подпись)

Преподаватель

\_\_\_\_\_ В.Э. Шмаков  
(подпись)

«\_\_\_\_\_» \_\_\_\_\_ 2023 г.

Санкт-Петербург  
2023

# Оглавление

<b>Лабораторная работа 1. Сокетные соединения</b>	<b>8</b>
1. Системные вызовы . . . . .	8
2. Присоединенные сокеты . . . . .	9
3. Локальные сокеты . . . . .	10
4. Интернет сокеты . . . . .	13
5. Модификация эхо-сервера . . . . .	14
Выводы . . . . .	16
<b>Лабораторная работа 2. Сокеты L4</b>	<b>17</b>
1. Анализ кода . . . . .	17
2. Компиляция и запуск . . . . .	17
3. Анализ состояния сокета . . . . .	18
4. Присоединенные сокеты . . . . .	18
5. Игра . . . . .	19
6. Состояние сокета в разные моменты . . . . .	20
7. Множество подключений . . . . .	20
8. Исправление игры . . . . .	21
Выводы . . . . .	22
<b>Лабораторная работа 3. Шифрование сообщений с помощью средств GNU Privacy Guard</b>	<b>23</b>

1. Установка . . . . .	23
2. Генерация ключей . . . . .	24
3. Шифрование и подпись текста . . . . .	30
4. Прочие возможности использования GPG . . . . .	33
Выводы . . . . .	33
<b>Лабораторная работа 4. Импорт и экспорт ключей. Цифровая подпись</b>	<b>34</b>
1. Экспорт открытого ключа . . . . .	34
2. Создание электронной цифровой подписи (ЭЦП) файла . . . . .	35
3. Импорт открытого ключа . . . . .	37
4. Проверка ЭЦП . . . . .	38
5. Экспорт/импорт на ключевые сервера . . . . .	39
Выводы . . . . .	40
<b>Лабораторная работа 5. Анализатор сетевого трафика Wireshark</b>	<b>41</b>
1. Установка . . . . .	41
2. Анализ ICMP трафика . . . . .	45
Канальный уровень (data link): Ethernet фрейм . . . . .	45
Сетевой уровень (network): IP пакет . . . . .	46
Сетевой уровень (network): ICMP пакет . . . . .	49
3. Анализ ARP трафика . . . . .	51
Канальный уровень (data link): Ethernet фрейм . . . . .	52
Сетевой уровень (network): ARP пакет . . . . .	53
3. Анализ FTP трафика . . . . .	54
Демонстрация уязвимости . . . . .	54
Шифрование трафика . . . . .	57
4. Сравнение защиты Telnet и SSH . . . . .	61

5. Анализ сообщений транспортного уровня: UDP-дейтаграммы и TCP-сегменты	64
Выводы	66
<b>Лабораторная работа 6. Аудит защищенности сети сканером Nmap</b>	<b>67</b>
1. Установка	67
2. Обнаружение хостов в сети	68
2.1. Наивное сканирование сети	68
2.2. Обнаружение компьютера методом ping	70
2.3. Обнаружение с помощью SYN/ACK- и UDP-пакетов	71
2.4. Обнаружение компьютера посредством различных ICMP-пакетов	74
3. Сканирование портов	75
3.1. Сканирование портов методом SYN	75
3.2. Сканирование с использованием системной функции connect()	77
3.3. Сканирование портов UDP-протокола	78
3.4. Сканирование с помощью методов FIN, Xmas и Null	81
3.5. Сканирование с помощью методов ACK и Window	82
3.6. Сканирование методом Maimon	83
3.7. Скрытое сканирование с использованием алгоритма idlescan	84
3.8. Сканирование на наличие открытых протоколов	84
3.9. Скрытое сканирование посредством метода ftp bounce	85
4. Изучение служб на удалённых хостах	86
4.1. Определение версии ОС	86
5. Параметры сканирования по расписанию	88
6. Сравните возможности Nmap с другими средствами аудита сети	89
Выводы	90
<b>Лабораторная работа 7. Утилиты Netcat и Cryptcat</b>	<b>91</b>

1. Установка . . . . .	91
2. Взаимодействие процессов . . . . .	92
3. Передача файлов . . . . .	94
3.1 Передача одного файла . . . . .	94
3.2 Передача одного файла с отображением прогресса . . . . .	95
3.3 Передача нескольких файлов . . . . .	95
3.4 Передача нескольких файлов со сжатием . . . . .	96
3.5 Передача образа жёсткого диска . . . . .	97
4. Защищенное взаимодействие . . . . .	97
4.1 Установка cryptcat . . . . .	97
4.2 Взаимодействие процессов . . . . .	98
4.3 Передача файлов . . . . .	99
5. Direct Network Traffic . . . . .	100
Выводы . . . . .	101

## **Лабораторная работа 8. Сетевое экранирование. Применение правил iptables 102**

1. Определение IP-адреса . . . . .	102
2. Просмотр текущие правил . . . . .	103
3. Блокировка входящего трафика . . . . .	103
4. Фильтрация входящего трафика . . . . .	105
Выводы . . . . .	107

## **Лабораторная работа 9. Сетевое экранирование. Работа с iptables 108**

1. Предусловия . . . . .	108
2. Запрет ICMP ping запросов извне . . . . .	108
3. Ограничение количества запросов . . . . .	110
4. Ограничение количества запросов . . . . .	111

5. Блокировка входящих запросов . . . . .	113
5.1 Блокировка по адресу . . . . .	113
5.2 Блокировка по порту . . . . .	114
5.3 Блокировка по адресу и порту . . . . .	115
5.4 Блокировка по MAC адресу . . . . .	115
6. Разрешение соединений только для протокола TCP . . . . .	115
Выводы . . . . .	120
<b>Лабораторная работа 10. Ограничение количества соединений</b>	<b>121</b>
Введение . . . . .	121
1. Файловые дескрипторы . . . . .	121
1.1 Ограничения файловых дескрипторов на уровне ядра . . . . .	122
1.2 Ограничения дескрипторов на уровне пользователя . . . . .	123
2. Процессы и потоки . . . . .	124
3. Параметры сетевого стека . . . . .	124
4. Ограничения IP Tables . . . . .	126
5. Ограничения запросов на уровне приложений . . . . .	126
5.1 Ограничение количества подключений (запросов) . . . . .	127
5.2 Ограничение скорости запросов . . . . .	130
5.3 Ограничение пропускного канала . . . . .	132
Выводы . . . . .	134
<b>Лабораторная работа 11. Сниффер заголовков сообщений протоколов уровней L2 и L3 модели OSI</b>	<b>135</b>
Введение . . . . .	135
1. Разработка приложения . . . . .	136
2. Демонстрация работы . . . . .	141

Выводы . . . . .	143
------------------	-----

# Лабораторная работа 1. Сокетные соединения

**Цель работы:** Освоение набора системных вызовов для создания сокетных соединений различных типов, для обмена данными на хостах и по сети.

## 1. Системные вызовы

**Задача:** Проанализируйте набор системных вызовов для серверной и клиентской сторон при организации соединений на сокетах под ОС Linux, принимая во внимание возможности различных видов сокетов и семейств адресации.

**Ход решения:**

Основные вызовы:

- `socket()` – создать новый сокет и вернуть файловый дескриптор;
- `send()` – отправить данные по сети;
- `receive()` – получить данные из сети;
- `close()` – закрыть соединение.

Основные вызовы на стороне сервера:

- `bind()` – связать сокет с IP-адресом и портом;
- `listen()` – слушает порт и ждет когда будет установлено соединение;
- `accept()` – принять запрос на установку соединения.

Основные вызовы на стороне клиента:



- `connect()` – установить соединение.

Основные семейства протоколов создаваемого сокета:

- `AF_INET` – для сетевого протокола IPv4;
- `AF_INET6` – для IPv6;
- `AF_UNIX` – для локальных сокетов (используя файл).

Основные типы соединений:

- `SOCK_STREAM` – надёжная потокоориентированная служба или потоковый сокет;
- `SOCK_DGRAM` – служба датаграмм или датаграммный сокет;
- `SOCK_RAW` – сырой протокол поверх сетевого уровня.

## 2. Присоединенные сокеты

**Задача:** Скомпилируйте и выполните программу `socketpair.cpp`, иллюстрирующую создание простейшего вида сокета и обмен данными двух родственных процессов.

Проанализируйте вывод на консоль. Существует ли зависимость обмена от различных соотношений величин временных задержек (в вызовах `sleep()`) в процессе-родителе и в процессе-потомке?

**Ход решения:** Беглый анализ исходного текста позволяет выявить следующие моменты:

1. В цикле `switch` дан не правильный комментарий для поведения по умолчанию: там сказано, что дальнейший код будет исполняться потомком, хотя это не так. Системный вызов `fork()` возвращает положительное не нулевое число для процесса-родителя.
2. Для организации сетевого взаимодействия используется системный вызов `socketpair()`, который создает пару безымянных присоединённых сокетов. Рассмотрим параметры, которые используются для этого системного вызова:
  - `PF_UNIX` показывает, что будет использовано локальное соединение.
  - `SOCK_STREAM` показывает, что семантика коммуникации обеспечивает создание двусторонних надежных и последовательных потоков байтов, поддерживающих соединения.

Остальные параметры тривиальны.

3. Учитывая, что потомок всегда пишет, и только потом читает, а предок действует наоборот, и при этом у нас блокирующие операции, сразу понятно, что будет простой поочерёдный обмен, и вызов `sleep()` ни на что не влияет. Или, другими словами, каждый этот вызов будет влиять и на одну и на другую сторону общения, увеличивая их ожидание.

**Эксперимент:** Запустив приложение, мы получили следующий вывод:

```
smart@thinkpad$ ./socketpair
p->c:0
c->p: 1
p->c:2
c->p: 3
p->c:4
c->p: 5
p->c:6
c->p: 7
p->c:8
c->p: 9
```

Дальнейшие эксперименты с `sleep()` подтвердили изначальные гипотезы: можно полностью убрать оба вызова `sleep()` и это не сломает программу, можно увеличивать значение параметра для `sleep()` и это будет влиять на оба процесса.

### 3. Локальные сокеты

**Задача:** Скомпилируйте программы `echo_server.cpp` и `echo_client.cpp`, задавая им при компиляции разные имена.

Запустите программы сервера и клиента на разных терминалах. Введите символьную информацию в окне клиента и проанализируйте вывод. Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия?

С чем связано создание специального файла в текущем каталоге во время исполнения программ?

**Ход решения:** Начнём с анализа исходных кодов.

Сервер:

1. Создаётся сокет, используя системный вызов `socket()`. Параметры `AF_UNIX` и `SOCK_STREAM` идентичны параметрам из предыдущего шага (`AF_UNIX` это синоним для `PF_UNIX`). Фактический результат этого вызова – файловый дескриптор.
2. Для соединения (`bind()`) сокета с адресом (`sockaddr_un`), производится подготовка этого адреса. Обычно тут задаётся адрес хоста и номер порта для ожидания соединения, но в данном случае адресу задаётся семейство, идентичное семейству сокета (`AF_UNIX`) и указывается путь, где сокет будет располагаться в файловой системе. Вызов (`unlink()`) позволяет автоматически удалить файл сокета, когда он перестанет использоваться.
3. После связывания, сокет переводится в режим прослушивания (`listen()`). Число 5 позволяет размер очереди клиентов, желающих подключиться.
4. В бесконечном цикле, сервер ожидает подключения клиента. Исполнение процесса будет заблокировано на вызове `accept()`. Этот вызов позволяет получить копию исходного сокета, чтобы слушающий сокет был готов к получению запросов от других клиентов.
5. После этого снова в бесконечном цикле происходит получение и отправка сообщений длиной в 100 символов через копию сокета, полученного на предыдущем шаге.

клиент:

1. Подобно серверу, создаётся сокет, используя системный вызов `socket()` с параметрами `AF_UNIX` и `SOCK_STREAM`.
2. На клиенте идёт подготовка к соединению. Обычно используется имя удалённого хоста и номер порта. Однако в нашем случае используется адрес сокета в файловой системе (адрес записывается в структуру `sockaddr_un`).
3. Далее следует непосредственно соединение (`connect()`).
4. В бесконечном цикле происходит считывание данных с устройства ввода (`stdin`), отправка их серверу, получение ответа от сервера и вывод этого ответа на стандартное устройство вывода (`stdout`).

**Эксперимент:** Запуск сервера:

```
smart@thinkpad$ ./echo_server
Waiting for a connection...
```

Запуск клиента:

```
smart@thinkpad$ ./echo_client
Trying to connect...
Connected.
> 123
echo> 123
> test
echo> test
> тест
echo> тест
> ~!@#$$%
echo> ~!@#$$%
>
```

После запуска сервера, в директории появляется файл сокета. Информацию о нём можно получить, к примеру, с помощью утилиты **ss**.

```
smart@thinkpad$ ss | grep echo_socket
u_str ESTAB      0      0          echo_socket 481033          * 481088
```

Эта запись означает следующее:

- **u\_str** – сетевой идентификатор;
- **ESTAB** – состояние соединения;
- **0** – количество пакетов в очереди на получение (**Recv-Q**);
- **0** – количество пакетов в очереди на отправку (**Send-Q**);
- **echo\_socket 481033** – локальный адрес и порт (большое число для локальных соединений);
- **\* 481088** – адрес и порт клиента.

## 4. Интернет сокеты

**Задача:** Скомпилируйте с разными именами программы `sock_c_i_srv.cpp` и `sock_c_i_clt.cpp` (в них используется общий `include` файл `local_c_i.h`). Запустите программы сервера и клиента на разных терминалах. При запуске клиента указывайте в качестве параметра командной строки имя хоста `localhost`. Введите символьную информацию в окне клиента и поясните вывод.

Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия?

**Ход решения:** Проведём анализ кода. Алгоритм практически аналогичен предыдущему примеру, но тут используется семейство `AF_INET` и создание отдельных процессов для обслуживания клиентов. В данном примере, мы как и раньше используем потоковую передачу (`SOCK_STREAM`), однако сетевые сокеты позволяют организовать передачу на датаграммах (`SOCK_DGRAM`) – такие соединения работают без подтверждения факта доставки (`ACK`).

И сервер и клиент при вызове функции `socket()` используют константу `AF_INET`, указывающую на то, что открываемый сокет должен быть сетевым. Сокеты в домене `AF_INET`, не знают про то, что они работают на локальной системе и обращаются только к `localhost`. Они полностью выполняют все механизмы сетевого стека: переключения контекста, `ACK`, `TCP`, управление потоком, маршрутизацию, разбиение больших пакетов и т.п. То есть это «полноценная `TCP` работа» несмотря на то, что пакеты не покидают локального интерфейса.

Дополнительные издержки при использовании `AF_INET` кроются так же в необходимости произвести резолвинг доменного имени в IP-адрес (вызов `gethostbyname()`) и решение проблемы little-big-end (вызов `htons()`, `htonl()`) связанной с разным порядком байтов на различных архитектурах (в комментариях кода написано что-то странное про "fake port").

Ещё одной особенностью является обработка подключений клиентов в отдельных процессах. После установки соединения, сервер вызывает `fork()` и работает с каждым клиентом отдельно. Это значит, что каждый клиент получит в ответ только свои сообщения.

**Эксперимент:** запуск сервера.

```
smart@thinkpad$ ./sock_c_i_srv
```

Запуск первого клиента.

```
smart@thinkpad$ ./sock_c_i_clt localhost
```

```
> 1
1
> 2
2
> 3
3
>
```

Запуск второго клиента.

```
smart@thinkpad$ ./sock_c_i_clt localhost
> a
A
> b
B
> c
C
> d
D
>
```

Замена строчных букв на заглавные происходит на стороне сервера при помощи команды `toupper(buf[i])`.

## 5. Модификация эхо-сервера

**Задача:** Модифицируйте программу `echo_server.cpp` так, чтобы при ответе на запросы клиента что-либо выводилось в окне сервера.

Испытайте работу эхо-сервера при работе с несколькими клиентами.

**Ход решения:**

Фактически, была добавлена только строка 3, представленная в листинге 1.

Листинг 1: Фрагмент исходного кода модифицированного файла (`echo_server_upd.cpp`)

```
1     if (!done)
2     {
```

```

3         printf("[client %i] %s",remSocket, str);
4         if (send(remSocket, str, textLength, 0) < 0)
5         {
6             perror("send");
7             done = 1;
8         }
9     }
10 } while (!done);

```

**Эксперимент:** запуск сервера.

```

smart@thinkpad$ ./echo_server_upd
Waiting for a connection...
Connected.
[client 4] 1
[client 4] 2
[client 4] 3
[client 4] 4
Waiting for a connection...
Connected.
[client 4] a
[client 4] b
[client 4] c
[client 4] d

```

Запуск первого клиента.

```

smart@thinkpad$ ./echo_client
Trying to connect...
Connected.
> 1
echo> 1
> 2
echo> 2
> 3
echo> 3
> 4
echo> 4

```

```
> ^C
```

Запуск второго клиента.

```
smart@thinkpad$ ./echo_client
Trying to connect...
Connected.
> a
b
c
d
echo> a
> echo> b
> echo> c
> echo> d
>
```

Наблюдаемое поведение полностью в рамках ожиданий: когда сервер установил соединение с первым клиентом, он находится в заблокированном состоянии на операции сетевого обмена. Как только первый клиент завершает свою работу, мгновенно происходит подключение и обслуживание второго клиента (он даже получает тот-же номер файлового дескриптора). Отсюда напрашивается вывод, что работа слушающего сокета должна производиться в одном потоке, а обслуживающих – в другом (других).

## Выводы

В данной работе мы познакомились с основным набором системных вызовов для создания соединений различных типов. Использование протоколов семейства `AF_INET` для локального подключения (`localhost`) оправдано только в том случае, если разработчик не знает откуда именно будет произведено подключение. Для взаимодействия в рамках одной системы следует предпочесть протоколы семейства `AF_UNIX` и избежать всей накладной работы связанной с сетевым стеком.



# Лабораторная работа 2. Сокеты L4

**Цель работы:** Создание клиент-серверных приложений, взаимодействующих друг с другом по сети на основе технологии соединения на сокетах L4.

## 1. Анализ кода

**Задача:** Проанализируйте код программы `server_game.cpp`, иллюстрирующей обмен данными с клиентскими приложениями по итеративной схеме.

**Ход решения:** Используется сокет семейства `AF_INET`. Сервер ожидает соединения на любом IP адресе (`INADDR_ANY`), но нам достаточно для подключения `localhost`. Для соединения будет использоваться порт 1066. Игра работает в один поток.

## 2. Компиляция и запуск

**Задача:** Скомпилируйте и запустите `server_game`.

**Эксперимент:** запуск сервера.

```
smart@thinkpad$ g++ server_game.cpp -o server_game
smart@thinkpad$ ./server_game
start to listen
```

Системный журнал зафиксировал выбор слова сервером.

```
smart@thinkpad$ journalctl -a | grep server_game
Feb 08 23:43:25 thinkpad server_game[103180]: server_game chose word
↪ green
```

```
Feb 08 23:43:40 thinkpad server_game[103180]: server_game chose word  
↪ green
```

### 3. Анализ состояния сокета

**Задача:** Запустите другой терминал и проверьте с него наличие в системе созданного сервером сокета и то, что он находится в состоянии **LISTEN**. Для этого выполните команду `netstat -a | grep 1066`.

Проанализируйте вывод данной команды и объясните ее смысл.

**Ход решения:** Так как команда `netstat` устарела и была заменена на `ss`, то будет рассматривать вывод `ss -atp | grep game`

**Эксперимент:** результат запуска команды `ss`

```
smart@thinkpad$ ss -atp | grep game  
State  Recv-Q Send-Q Local Address:Port      Peer Address:PortProcess  
LISTEN 0      5          0.0.0.0:fpo-fns      0.0.0.0:*  
↪ users:(("server_game",pid=103180,fd=3))
```

В результате мы видим, что в данный момент сокет находится в состоянии **LISTEN**. В очереди на получение находятся 0 пакетов, на отправку 5. Сервер слушает на любом адресе, но вместо номера порта мы видим `fpo-fns`. Порт 1066 зарезервирован в системе для какой-то компьютерной игры, поэтому мы видим название вместо номера. Дальше мы видим, что клиент пока не подключился. А в самом конце строки общую информацию о процессе.

### 4. Присоединенные сокеты

**Задача:** Запустите в качестве клиентского процесса утилиту `telnet` с параметрами: `telnet localhost 1066`. При организации коммуникации по сети на разных компьютерах вместо `localhost` при запуске клиента указывается IP-адрес компьютера, на котором был запущен сервер.

**Ход решения:** Ввиду устаревания `telnet`, воспользуемся утилитой `gnu netcat`.

**Эксперимент:** Запуск клиента

```
smart@thinkpad$ nc localhost 1066
:laying on host: h
```

Программа запустилось, соединение установлено. Но имя хости на сервере ничем не инициализированно.

## 5. Игра

**Задача:** Диалог с сервером заключается в угадывании слова. Оно вводится по буквам с клиентского терминала. При этом сервер вместо неугаданных букв выдает символы ”-”, а также считает число оставшихся неудачных попыток (всего их предусмотрено 12).

**Эксперимент:** Запуск клиента

```
smart@thinkpad$ nc localhost 1066
:laying on host: h

----- 12
ф
----- 11
g
g---- 11
r
gr--- 11
e
gree- 11
e
gree- 11
b
gree- 10
n
green 10

green 9
```

Заметно, что логика игры не доделана: победная ситуация не обрабатывается корректно.

## 6. Состояние сокета в разные моменты

**Задача:** Завершите серверное приложение с помощью сигнала `kill`, и затем определите командой `netstat -a | grep 1066`, когда исчезает из системы соединение на сокетах. Во время сеанса обмена также примените команду `netstat -a | grep 1066`, чтобы исследовать состояние соединения.

**Эксперимент:** Состояние сокета после завершения игры

```
smart@thinkpad$ ss -atp | grep game
```

State	Recv-Q	Send-Q	Local	Address:Port	Peer	Address:Port	Process
-------	--------	--------	-------	--------------	------	--------------	---------

Сокет не обнаружен.

Состояние сокета в процессе игры

```
smart@thinkpad$ ss -atp | grep game
```

State	Recv-Q	Send-Q	Local	Address:Port	Peer	Address:Port	Process
LISTEN	0	5		0.0.0.0:fpo-fns		0.0.0.0:*	
↪ users:(("server_game",pid=109301,fd=3))							
ESTAB	0	0		127.0.0.1:fpo-fns		127.0.0.1:52880	
↪ users:(("server_game",pid=109301,fd=4))							

Как и ожидалось, мы видим один слушающий сокет, и один установленный. При этом `127.0.0.1:fpo-fns` – данные сервера, а `127.0.0.1:52880` – данные клиента.

## 7. Множество подключений

**Задача:** Прodelайте все заново, но запускайте не одно клиентское приложение (в виде `telnet`), а несколько экземпляров с разных терминалов, и попытайтесь работать с них одновременно.

Проанализируйте, как сервер будет обслуживать запросы в этом случае.

**Эксперимент:** После запуска двух дополнительных клиентов, мы видим, что они находятся в состоянии блокировки на операции `connect`, и не получают сообщений от сервера.

```
smart@thinkpad$ ss -atp | grep game
```

State	Recv-Q	Send-Q	Local	Address:Port	Peer	Address:Port	Process
-------	--------	--------	-------	--------------	------	--------------	---------

```

LISTEN 2      5      0.0.0.0:fpo-fns      0.0.0.0:*
↪ users:(("server_game",pid=109301,fd=3))
ESTAB 0      0      127.0.0.1:fpo-fns      127.0.0.1:52880
↪ users:(("server_game",pid=109301,fd=4))

```

Состояние сокета показывает очередь на слушающем сокете длиной в 2.

## 8. Исправление игры

**Задача:** Модифицируйте программу `server_game.cpp` так, чтобы запросы от каждого из клиентов могли обслуживаться конкурентно, путем запуска для каждого нового соединения собственного нового процесса на сервере или потока. Проанализируйте, как обслуживаются запросы в случае конкурентной схемы работы сервера.

Возможно также улучшить качество самой игровой функции `guess_word()` сервера.

### Ход решения:

Обработка подключений производится в отдельном процессе

Листинг 2: Фрагмент исходного кода модифицированного файла игры

```

1  if (fork() == 0)
2  { /* In child process */
3      guess_word(fd, fd);
4  }

```

Исправлены условия для определения победы

Листинг 3: Фрагмент исходного кода модифицированного файла игры

```

1  char word_[MAXLEN];
2  word_[0] = 'g';
3  word_[1] = 'r';
4  word_[2] = 'e';
5  word_[3] = 'e';
6  word_[4] = 'n';
7  word_[5] = '\0';

```

### Эксперимент:

Состояние сокета показывает успешную работу с тремя клиентами.

```

smart@thinkpad$ ss -atp | grep game
State  Recv-Q Send-Q Local Address:Port      Peer Address:PortProcess
LISTEN 0      5            0.0.0.0:fpo-fns        0.0.0.0:*
↪ users:(("server_game",pid=113319,fd=3),("server_game",pid=113317,fd=
↪ 3),("server_game",pid=113310,fd=3),("server_game",pid=113306,fd=3))
ESTAB  0      0            127.0.0.1:fpo-fns      127.0.0.1:52976
↪ users:(("server_game",pid=113310,fd=4))
ESTAB  0      0            127.0.0.1:fpo-fns      127.0.0.1:52978
↪ users:(("server_game",pid=113317,fd=4))
ESTAB  0      0            127.0.0.1:fpo-fns      127.0.0.1:56378
↪ users:(("server_game",pid=113319,fd=4))

```

Победная ситуация успешно обрабатывается, имя хоста определяется правильно.

```

smart@thinkpad$ nc localhost 1066
Playing on host: thinkpad:

----- 12
g
g----- 12
r
gr--- 12
e
gree- 12
n
green 12
You ween! Congrats!

```

## Выводы

В данной работе мы выполнили практическую имплантацию взаимодействия клиентского и серверного приложения в рамках реализации игры с угадыванием слов.

В практическом плане, такой подход может быть небезопасен: порождение процесса на каждого клиента достаточно дорогостоящая (в плане производительности) операция, а неконтролируемое их порождение может привести к атаке типа DDoS.

# Лабораторная работа 3. Шифрование сообщений с помощью средств GNU Privacy Guard

**Цель работы:** Знакомство с возможностями утилиты GNU Privacy Guard.

## 1. Установка

Установка утилиты не потребовалась, т.к. она уже установлена на компьютере.

```
smart@thinkpad$ pacman -Qi gnupg
Name           : gnupg
Version        : 2.2.40-1
Description    : Complete and free implementation of the OpenPGP
↳ standard
Architecture   : x86_64
URL            : https://www.gnupg.org/
Licenses       : BSD  custom  custom:CC0  GPL2  GPL3  LGPL3  LGPL2.1
↳ MIT
Groups         : None
Provides       : None
Depends On     : bzip2  libbz2.so=1.0-64  glibc  gnutls  libgcrypt
                  libgpg-error  libksba  libassuan  libassuan.so=0-64
↳ npth
                  libnpth.so=0-64  pinentry  readline
↳ libreadline.so=8-64
                  sqlite  zlib
Optional Deps  : libldap: gpg2keys_ldap [installed]
```

```
libusb-compat: sddaemon
pcsc-lite: sddaemon [installed]
Required By      : gpgme  pacman  visual-studio-code-bin
Optional For     : None
Conflicts With   : None
Replaces         : None
Installed Size   : 8.55 MiB
Packager         : David Runge <dvzrv@archlinux.org>
Build Date       : Fri 14 Oct 2022 02:01:35 PM MSK
Install Date     : Mon 19 Dec 2022 11:46:09 PM MSK
Install Reason    : Installed as a dependency for another package
Install Script    : Yes
Validated By     : Signature
```

Информация о текущей версии GnuPG и поддерживаемых криптоалгоритмах

```
smart@thinkpad$ gpg2 --version
gpg (GnuPG) 2.2.40
libgcrypt 1.10.1-unknown
Copyright (C) 2022 g10 Code GmbH
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/smart/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

## 2. Генерация ключей

Произведём генерацию ключей.



Для диалога генерации ключа я использую флаг `-full-generate-key`, т.к. стандартный `-gen-key` скрывает некоторые вопросы диалога используя значения по умолчанию (в частности, используется ключ длиной 3072).

```
smart@thinkpad$ gpg2 --full-generate-key
gpg (GnuPG) 2.2.40; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
  (1) RSA and RSA (default)
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 3w
Key expires at Sat 04 Mar 2023 05:10:43 PM MSK
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Semen Martynov
Email address: martynov.sa@edu.spbstu.ru
Comment: Test digital sirnature
You selected this USER-ID:
    "Semen Martynov (Test digital sirnature) <martynov.sa@edu.spbstu.ru>"
```

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

gpg: /home/smart/.gnupg/trustdb.gpg: trustdb created

gpg: directory '/home/smart/.gnupg/openpgp-revocs.d' created

gpg: revocation certificate stored as '/home/smart/.gnupg/openpgp-revocs.d/9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6.rev'

public and secret key created and signed.

pub rsa4096 2023-02-11 [SC] [expires: 2023-03-04]

9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6

uid Semen Martynov (Test digital sirnature)

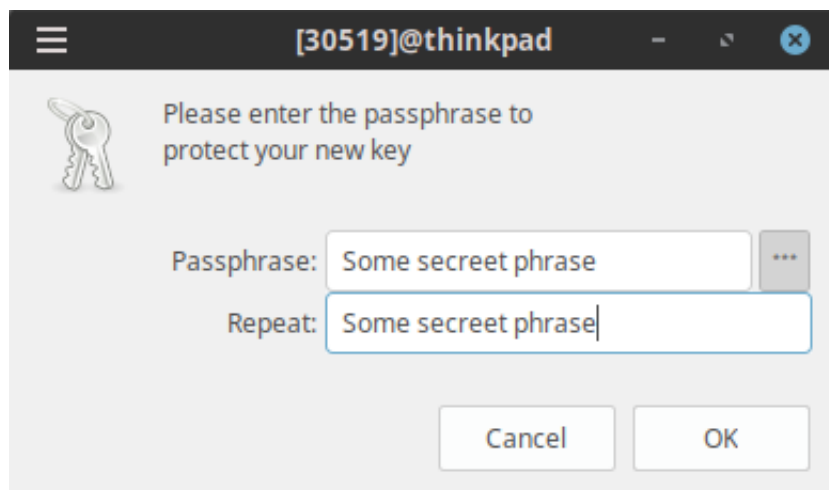
↪ <martynov.sa@edu.spbstu.ru>

sub rsa4096 2023-02-11 [E] [expires: 2023-03-04]

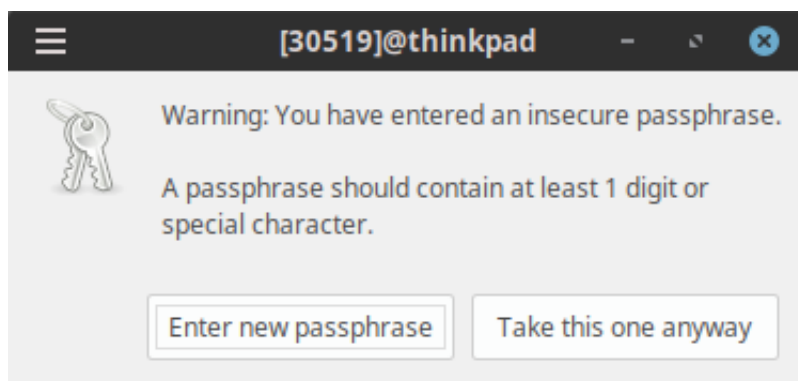
В режиме диалога, были выбраны следующие опции

- Использование ключа с двумя типа шифрования (RSA и RSA)
- Длина ключа в 4096 бит (с 2015-го года NIST рекомендует использовать ключи длиной в 2018 бит, но уже сейчас многие компании перешли на 4096)
- 3 недели в качестве срока жизни ключа (как долго он будет валиден для использования)
- Имя и e-mail предоставленный университетом.

Дальше выполнение диалога прервалось для ввода ключевой фразы, защищающей ключи от несанкционированного использования.



Был выбран слабый пароль для шифрования, о чём система меня предупредила. Продолжим с небезопасным паролем.



Текущий список ключей с keygrip (идемпотентный протокол хеширования для связи ключей)

```
smart@thinkpad$ gpg --list-keys --with-keygrip
/home/smart/.gnupg/pubring.kbx
-----
pub   rsa4096 2023-02-11 [SC] [expires: 2023-03-04]
      9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
      Keygrip = EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4
uid           [ultimate] Semen Martynov (Test digital sirnature)
↳ <martynov.sa@edu.spbstu.ru>
sub   rsa4096 2023-02-11 [E] [expires: 2023-03-04]
      Keygrip = 16E5EFCBA47102BBA3A7C17D467D430C13D3D43E
```

- pub – публичный ключ (keygrip EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4);

- uid – идентификатор (User-ID);
- sub – публичный подключ (keygrip 16E5EFCBA47102BBA3A7C17D467D430C13D3D43E);

#### Список приватных ключей

```
smart@thinkpad$ gpg2 --list-secret-keys --with-keygrip
/home/smart/.gnupg/pubring.kbx
-----
sec    rsa4096 2023-02-11 [SC] [expires: 2023-03-04]
      9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
      Keygrip = EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4
uid          [ultimate] Semen Martynov (Test digital sirnature)
↳ <martynov.sa@edu.spbstu.ru>
ssb    rsa4096 2023-02-11 [E] [expires: 2023-03-04]
      Keygrip = 16E5EFCBA47102BBA3A7C17D467D430C13D3D43E
```

- sec – секретный ключ (keygrip EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4);
- uid – идентификатор (User-ID);
- ssb – секретный подключ (keygrip 16E5EFCBA47102BBA3A7C17D467D430C13D3D43E);

#### Список подписей

```
smart@thinkpad$ gpg2 --list-signatures --with-keygrip
/home/smart/.gnupg/pubring.kbx
-----
pub    rsa4096 2023-02-11 [SC] [expires: 2023-03-04]
      9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
      Keygrip = EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4
uid          [ultimate] Semen Martynov (Test digital sirnature)
↳ <martynov.sa@edu.spbstu.ru>
sig 3      A0B01E0BAB2AF7C6 2023-02-11  Semen Martynov (Test digital
↳ sirnature) <martynov.sa@edu.spbstu.ru>
sub    rsa4096 2023-02-11 [E] [expires: 2023-03-04]
      Keygrip = 16E5EFCBA47102BBA3A7C17D467D430C13D3D43E
sig      A0B01E0BAB2AF7C6 2023-02-11  Semen Martynov (Test digital
↳ sirnature) <martynov.sa@edu.spbstu.ru>
```

Отпечаток (fingerprint) подписи

```
smart@thinkpad$ gpg2 --fingerprint martynov.sa@edu.spbstu.ru
pub   rsa4096 2023-02-11 [SC] [expires: 2023-03-04]
       9DAF 94FC D9CA 38BF D298 BDOC A0B0 1E0B AB2A F7C6
uid           [ultimate] Semen Martynov (Test digital sirnature)
↳   <martynov.sa@edu.spbstu.ru>
sub   rsa4096 2023-02-11 [E] [expires: 2023-03-04]
```

Исследуем содержимое директории .gnupg

```
smart@thinkpad$ tree .gnupg/
.gnupg/
  openpgp-revocs.d
    9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6.rev
  private-keys-v1.d
    16E5EFCBA47102BBA3A7C17D467D430C13D3D43E.key
    EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4.key
  pubring.kbx
  pubring.kbx~
  trustdb.gpg
```

В директории можно увидеть

- `openpgp-revocs.d/9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6.rev` – предварительно созданный сертификат отзыва. Имя файла соответствует отпечатку ключа. Всякий, у кого есть доступ к этому файлу, может отозвать соответствующий ключ.
- `private-keys-v1.d/EC3AFAA6FD0AE71FB449D7D42849E173CF6CAAF4.key` – приватный ключ.
- `private-keys-v1.d/16E5EFCBA47102BBA3A7C17D467D430C13D3D43E.key` – приватный подключ.
- `pubring.kbx` – Таблица открытых ключей.
- `pubring.kbx~` – Таблица открытых ключей (резервная копия).

- `trustdb.gpg` – База данных доверия (Алиса подписала публичный ключ Боба, а Боб подписал публичный ключ Чарли; если Алиса получит публичный ключ Чарли, она сможет ему доверять, потому что ключ подписан тем, кому Алиса доверяет, т.е. Бобом).

### 3. Шифрование и подпись текста

Для демонстрации возможностей утилиты, сгенерируем простой Lorem Ipsum.

Листинг 4: Lorem Ipsum

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean in accumsan
  ↪ tellus. Quisque et sem sodales, rhoncus dolor ac, hendrerit augue.
  ↪ Donec elementum vitae metus at lobortis. Mauris facilisis nisl et
  ↪ viverra facilisis. Phasellus nec dolor nunc. Nullam euismod, quam ut
  ↪ consequat mattis, magna orci pulvinar ipsum, tincidunt dapibus erat
  ↪ nibh nec neque. Duis non elit at lacus tempus finibus. Nam tincidunt
  ↪ rutrum odio. Vivamus et urna a ante condimentum faucibus sit amet in
  ↪ tellus. Maecenas euismod erat ac dignissim vestibulum.

Nunc magna leo, ultricies vel ipsum vel, feugiat mollis turpis. Duis gravida
  ↪ eros vitae nisl faucibus, et varius mauris convallis. Curabitur at
  ↪ dui in risus sagittis consectetur id eu ex. Suspendisse posuere, quam
  ↪ eget aliquet efficitur, dolor nisi finibus ipsum, in hendrerit sem
  ↪ orci eu ante. Aliquam volutpat tellus id ante scelerisque, vel
  ↪ faucibus ante interdum. Mauris rutrum ante elementum magna aliquet
  ↪ finibus. Praesent mattis libero sit amet egestas tincidunt. Sed
  ↪ pretium urna bibendum iaculis dignissim. Sed id volutpat risus, eu
  ↪ feugiat arcu. Pellentesque a massa ac neque molestie lacinia eu sed
  ↪ elit. Morbi vel tempus odio, sit amet vehicula odio.
```

Теперь зашифруем файл с тестовым выводом

```

smart@thinkpad$ gpg2 \
  --armor \
  --recipient 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \
  --encrypt LoremIpsum.txt
```

Листинг 5: зашифрованный Lorem Ipsum

```

-----BEGIN PGP MESSAGE-----
```

```
hQIMA1btWtwni6bhAQ//QuiTJLY/3YvGJymWXLrJLW/YkuuSD6EUKPj0FGLnTNJp
lIblym8wePclTScHWTwf98d0j5hQKsJQaEf5EgGQ11B1/7mGAzL113QDcJJKqG3N
OWcc7kbjaJJZK3LyXe8ALT+HgiNAY7Yr8r/TP5vFoepwd60MVnxA+d8HD31gFP9
P9sZLlsgxvoQzQWsxjVG4mS1MJssznGJ9YA5ytz/Oc1asE+CYOuTaNk7zKoybWKd
oZvKNe7vlszf5f/TM3zVwasj9M/Di/cZX40utjustaz7lQrYGUp47p0rhYLbus1I
1BgH9WcoM7+2odTfkK0/ufhD0yNaSSJBvfJ+erqk9oZs7Jw0Woef8vqi89bMf9vh
yhlWoXyf5gyLRb+71leS/qeS017CDeUHTSR8Ac3LHc35gxRUaxWK/ID7/zN3+MnZ
AJvxyMUXyvSCWNERhLKB/b4v633mAreq9/5+crw2ZCyaZ90BaXc6IzqzCE25Fn87
OvmTS5gGITGlbYRwxKFijcgp7GEtjg6ywnvBq35eKvNKRkuCKPXuT5+00Buyi3s8
44wiC/Jft2SPH53P7QJd7ef7bC2e8+uojDbtVA7H5dxA780j1t1btCCdNG6WcMEg
Vr+K//WxQ1sroWrvx/u4mNHSdQEMy2Mojsgp5DndfIFq7DSqWsmihsG5qTGoW7HS
6QGAGa8BYCPSBp4shb+NSAkMySyHnCN4qKSeSPm2yvgyRbbSYv0Z2/f2f9jwyY8r
9m9eCym4wZ9ccr0pFU9pc37drxhizWuhLaUWJqtCL7vxlWmenJoXT1tFckF14g9s
1+GreN3hAk+fckEtE6rkq6wFiXJINULxkDBPIwPgaZMHv5ELhdhEl+Wu/b4Nhgz
m1zxe6Hw2aKTHuJaTr9xtYrSgXEH46CvHxYzT8m3bTevyEsSY/WqhnzVP3w/bb4a
sGo5BkRgD1UME/fhjk03v0NUVzFXzNwu1S650XqmUfzoQdElSq001B5m3MHa/3si
tkTpsn4DFlt6DVsmoi91mJ9e9Z939oIdc8z63a4kDPnQtbuqNbGQs4HklJmqe6Cw
cNi/FmceiEhBYJWliX8ahJSQTAsAXwU0eOiNj1tbD7AKm77DFgPn9enwB4sDLhqc
/feiT+x/seZ6/SXp+AU766YR8oXnndxmpdTf59UKACn9vm+eOblmMvXcr/K1LpGC
WrnTmSrCb8MvuIKguuRStmiTLWfzLlTML84DS+aDQbmKGLwtqCPYxJVgOkDnUiq9
5y0zPlkW1pry87Iujl+ljbPXWGGyz9uDlfrhtYngeewqw5l2aaCytSFujKup6lN0
ojvRN15wzvlCXgBy+oSOW4urv9IE1eaXOKRDCm4nZXq0rLhCx6uKnzAYgLjbHby9
RuBAIE5pD/CLJKmw0nW24FnAR7BcQCqul/Zv0QC+W2i89MiPpHgaXXgKydh8isp
FoLDG6hya0cAaKaL2jBquct3ggq6rrabw84XhDsgevXr0qj94mSCKuBY/BUHJ8xN
eTda/MWeSUuy7XP7zCBA1bPpU1eQjCvkCLqe5b49v1S8A8Gaep0t/+UieBm3a1G0
Y8NkaBfu01NjnkaHH0o=
=pkz1
-----END PGP MESSAGE-----
```

Далее расшифруем файл (операция потребует ввода приватной фразы!)

```
smart@thinkpad$ gpg2 \
--recipient 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \
--decrypt LoremIpsum.txt.asc > LoremIpsum.decrypted.txt
```

Убедимся, что расшифрованный файл соответствует исходному.

```
smart@thinkpad$ shasum -a 1 LoremIpsum.txt LoremIpsum.decrypted.txt
014bbacd9d092d1f2272a3419125dbb2de0f3a6b  LoremIpsum.txt
014bbacd9d092d1f2272a3419125dbb2de0f3a6b  LoremIpsum.decrypted.txt
```

```
smart@thinkpad$ gpg2 \
```

```
--recipient 9DAF94FCD9CA38BFD298BDOCA0B01E0BAB2AF7C6 \  
--clearsign LoremIpsum.txt
```

### Листинг 6: Lorem Ipsum с цифровой подписью

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA256  
  
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean in accumsan  
    ↳ tellus. Quisque et sem sodales, rhoncus dolor ac, hendrerit augue.  
    ↳ Donec elementum vitae metus at lobortis. Mauris facilisis nisl et  
    ↳ viverra facilisis. Phasellus nec dolor nunc. Nullam euismod, quam ut  
    ↳ consequat mattis, magna orci pulvinar ipsum, tincidunt dapibus erat  
    ↳ nibh nec neque. Duis non elit at lacus tempus finibus. Nam tincidunt  
    ↳ rutrum odio. Vivamus et urna a ante condimentum faucibus sit amet in  
    ↳ tellus. Maecenas euismod erat ac dignissim vestibulum.  
  
Nunc magna leo, ultricies vel ipsum vel, feugiat mollis turpis. Duis gravida  
    ↳ eros vitae nisl faucibus, et varius mauris convallis. Curabitur at  
    ↳ dui in risus sagittis consectetur id eu ex. Suspendisse posuere, quam  
    ↳ eget aliquet efficitur, dolor nisi finibus ipsum, in hendrerit sem  
    ↳ orci eu ante. Aliquam volutpat tellus id ante scelerisque, vel  
    ↳ faucibus ante interdum. Mauris rutrum ante elementum magna aliquet  
    ↳ finibus. Praesent mattis libero sit amet egestas tincidunt. Sed  
    ↳ pretium urna bibendum iaculis dignissim. Sed id volutpat risus, eu  
    ↳ feugiat arcu. Pellentesque a massa ac neque molestie lacinia eu sed  
    ↳ elit. Morbi vel tempus odio, sit amet vehicula odio.  
-----BEGIN PGP SIGNATURE-----  
  
iQIzBAEBCAAAdFiEEna+U/NnKOL/SmLOMoLAeC6sq98YFAMpNyRAACgkQoLAeC6sq  
98acig//Rusl7Np80tyIPyDCgycoe0irJX64r/P6ggAa1LyMb0FwVbx29pD+gkDy  
4oq8mcXGW07elL7MrOG7WEpjOnURpGcyeYcrZGodSFCPl841rGtOtZo61w2tI27k  
omNjCUEEG64Bc//LYe8nk7vxQdxRFS3sH0M0mfKoI7hLzuThZZkrE64pnZbulMN  
hdab7W8tDv/mZ2iMl0IoZnEb23zF+dEEpoMQ4R6xzV03jqwBso+QshKN8iQYkXG  
kk1Ll3zvLRgQbnRoX7uaHwET9ENuhdx1Y/eGHJ6slkxspmeCk0sYoph7eB6qf3AS  
XdKRHGtvRKsF+6YBMUcjre4XI09Ruf8+UvH5DldJZ8rhS5xxZQPiVt9WUxLlTYAn  
f0qsIKpciRcTWVCGZWsCtb+EDxIGwKSk08/luvTB+q27+oH6OWlt2mcAuEuoCLBp  
/wzUaA7dikmLdUbcig3hibFF+PnmOhkvqSNPNPIdsmGoYa86h8HINFI mugtHgdUd  
9Fq/4E6LSFALcV87xsuQ/073o0PHvKAeWt10EhJDYlJjIUKoU5UirZFU0+qiH/9D  
AWzJjJRPqG9wkM1j4uY5pmGk9YgWJerS5t7b8HXfdi8w5snbvtkbSiP81Ku/5ior  
pcf9GfD8rGe143dpx9GMPGABF4+pGis/i8h5VjgISFZ7XKQdevQ=  
=LzRD  
-----END PGP SIGNATURE-----
```

Проверка цифровой подписи



```
smart@thinkpad$ gpg2 --verify LoremIpsum.txt.asc
gpg: Signature made Sat 11 Feb 2023 07:57:52 PM MSK
gpg:                using RSA key
↳ 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
gpg: Good signature from "Semen Martynov (Test digital sirnature)"
↳ <martynov.sa@edu.spbstu.ru> [ultimate]
```

## 4. Прочие возможности использования GPG

Популярным способом использования GPG является цифровая подпись коммитов в git. Для этого в `.gitconfig` нужно добавить следующие параметры

```
[commit]
    gpgsign = true
[user]
    signingkey = <KeyID>
[gpg]
    program = /bin/gpg
```

## Выводы

В данной работе мы провели знакомство с возможностями утилиты GNU Privacy Guard. Разобрались с процессом генерации ключей и их хранением. Выполнили шифрование и сделали цифровую (присоединенную) подпись документа.

В практическом плане, использование GNU Privacy Guard позволяет подписывать коммитов в git для однозначного установления авторства.

# Лабораторная работа 4. Импорт и экспорт ключей. Цифровая подпись

**Цель работы:** Знакомство с возможностями утилиты GNU Privacy Guard.

## 1. Экспорт открытого ключа

Экспорт открытого ключа в текстовый файл

```
smart@thinkpad$ gpg2 \  
  --armor \  
  --export 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 > mykey.asc
```

Листинг 7: Открытый ключ mykey.asc

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
  
mQINBGPnogsBEADSTr64EalS52nYbQpCiUJHDinPA+UB3wkg8gfB5xx966DUtx0J  
P8++XKCqrbTLvxgdtLtSu4KlATC+xlj9RY0VuPXBpA/BgCArG1nI9GfG/fKqKLnl  
v2ETMTIQgLRzmiN/pZNidsi4kZN1KUeE7lD0/hH9LVcSNtB4tNvOscdASj9S8m58  
yoFqgDMhkAu0l2MyowzJSeRlAEdXkkoWlrxAvElUrqU8kwD6gWjiB4j+jWEVSqC  
cfl3XaCGFJ8VDZ5P9sVDXl83a4GnDZG1VhzmW0ZaBynacLKFpsbaIK9JvsNS0bN6  
aWK9G9yY/y9SEzWupDDH/eb4gCX9CZkuIFqwaQpOetnKNm6D2k9XgD80NYbFVkyP  
YjJJJytgjtKDr/TQ0ouGu5lfi2XBMvzXoDCd4Nj/VBKkGH92InNXCdP078yRZTzJ  
l787ULxPYXRv0sagezsIkL8vyQWei1W/CZ95271w90MBpUpozAUSfhiWEIcLvsv  
nNLksMZ93ELmRdMBppU3d/IAyLQsnTudL61amr89kRSVsmUY5i1JnvWIm3LyVcvA  
DcgceSJGQ3unCtvez7X3r85SxJSEWBhC8KQPGVq5DJy/0XgFvar5Yqg3ZDr909r5  
IerygJ0aoSAvGz3GDJZvqF1tKhTesm415+bD7xrJHljz1mfPhS0Zb+docwARAQAB  
tENTZWl1biBNYXJ0eW5vdiAoVGZvdCBkaWdpdGFsIHNPcm5hdHVyZSkgPG1hcnR5  
bm92LnNhQGvkdS5zcGJzdHUucnU+iQJUBBMBCAA+FiEEa+U/NnKOL/SmLOMoLAe  
C6sq98YFamPnogsCGwMFCAbr4AFCwkIBwIGFQoJCAasCBBYCAwECHgECF4AACgkQ  
oLAeC6sq98apHw//fQa/KFEIBrNNZWuXEL5ds3R3Ty4M81niKs6Zpdgvw90MFbVn
```

```
zLH5S0XSDm1Qln3r0Zo77GiVnbt1dbPpY6Xn1CkxqXKTmZBAkIEINyQ37II27UG5
1qosIcxVTxkEPe6HGJbJavGcxXh57upr1FyuSdEbJq0QiC3mJQGsJUt3L68m8ygR
hUu83xbRQmCHqhCFD/0G9d5C0s3WgNvMe8KuiEQYR8UELHnQDy6+Me32pfG8kXwN
DS0D/wbdMX9MFudsRivtHX01fY3ynZoEEEZo3c+jS+xszm0cMXXJsB57uBrHl3+d
p/v1snysyukhKBRT4RULn9vEIy0wkSlzlE3tyicnYzT2icjp9+DFzuVsx+w/mosY
7fWZAqdL6dol3+32V9jLG58v1G03odb6hY0yzc+tEDKugX9mRWk9ew+zaJQw57fq
yzlHszXNWKrsbf9wtiKC7ycuMfsDmYUUmUAYmFdev6D3gZNb8u5fe+PosDbhV/K/
3/000BD2DHWjhFUPKuKLwPiX5gENY5u4DIoe2uKJl0MSrqUDJW7Wm+ieqX9EwekJ
QDTzP0gA0GqQ0zpBKlMaW1s8FVwGbGRj/g7xENzx5Dqds7Rn8BFz++smiyGEzawtI
m5U9iQwE0rplrCtJUJ0QdIJM1GBcDV4GW3l6NlFB98lasEycsFnKpUanKGW5Ag0E
Y+eiCwEQAL0Vn+PMAS99bgLZVvW5p2a98Y4dkKbkypXsAT5mXTxmbfHdXYJ/rTJw
MBdFPoAX6bRnpnvd0ApDaLF7wrDXsHIZ4BW/dEyE8wwfui4yiGS3hHAM50AEh1I4
kLn4Ie2MUssx4j9SQr0IZOSSfGZuYConHSWe0xGETJerkhfn4PvymZ9j4yoCO/x+j
NTep4g6m+q13m/sBXo9600D+jbLr24zgyQcHKYIk6UHD62P9oUvLLfeD86yKlCUB
AosbesJN04JQTTZNunGDn1IiIthG10jeWF1aXN0x4q2LthG0sZbY0Vpz/Fwpl9Jq
RmhsCkp/7+CTi0fvcvGiytq3mk86hGC1qCHsVluyqNOW4WD9epyq9pnGRjSmTdj2
s5LwuZM61iqpkelk19s8T6o5Tz7j6ZpyTCnb6ZYwqrcao2Zqye3oMgbHYzWXI+t8
7AR8hYPdUYAks7m1KCSeFYPZxQVtwE4v/V+nm93f8hWvUG8LaLnsJ2Me2hb1CZLC
3R/b86PW9ZgAYH6JS0XbQJyLDZqCYpbJyCihY+fHPJV8z3U7POVIZLGxvimdB4bC
9eFNXcuAaXRFWWUVw2sHYYYaLg1moiSzfXm7V01k8/1b8urMVi5tXanr2y4pZqS7
MZ3tCERZLonSNu08R0nvpAalUX07gWHg6tYg/Z2e4uyt+VTZSb4VABEBAAGJAjwE
GAEIACYWIQSdr5T82co4v9KYvQygsB4Lqyr3xgUCY+eiCwIbDAUJABuvGAAKCRcg
sB4Lqyr3xj5BD/4xVupEmhDxKuQ6eC0UnZhcnI9fPfj7ULD0LCQ0mcNPmkAk7FqN
HRm7bp8Anz6JMG+bh1flyt/hiJUaqLvq9Mxz4B50ZohTrjqDPBRYyxtlABN/rAp
ZtdeUyagZXndUlbKFZDIdZjacpYRID3jC3rFYSQBky7j96hh3PzER7ld5JZ1r5YW
6/bJNu0meRkzBc9ggGopYjPUPpPjmwalmN4ZFMH8I1lj0ma8Rj089CzXnQnNfjHT
9knF7P1480ZP71s7ZpLDLjtqTCemmP/pa9ia020PjXjPXlmnkuqKSv9/bMa3bMaQ
J4CYBIHYo98KW0xt1Iw589NZT8MDhtcHb2gmdbDqRg1nNKEjN7TxZ33zLNSK0auM
I7NqZuXUZnrPP5o0WhN6RjPqyB/02MDNvhTCWhicf364S4d+D9owf1dMOHZS9XLp
s4NLGq+7p8M9g6W+jKWVqhDa0PTzyAJuWn7N0pszbQx1zRRH0d+TRpTS0uQawwmD
obhXjes/nZlFZMkLzuBopyrt6H4xrfJ7J46KziBdAuFAh14EMcaA/x+Y5yjTG7t3
HwqYttvuCffdnxfpbx+xW95LjCdWa+ba1h55RoNPakseGHVspxDKLyI2A8fDB69Mc
uQIpX0XSbCGB6cdi/t/9u9eRLg966CSmZ6vDHaAkukJadKAu13axbhh6cg==
=mu5o
-----END PGP PUBLIC KEY BLOCK-----
```

## 2. Создание электронной цифровой подписи (ЭЦП) файла

Отсоединённая ЭЦП файла mydocument.pdf в текстовом формате (команда требует ввода ключевой фразы)

```
smart@thinkpad$ gpg2 \
--sign \
--detach-sign \
--default-key 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \
--armor mydocument.pdf
```

Листинг 8: Отсоединённая цифровая подпись для файла mydocument.pdf

```
-----BEGIN PGP SIGNATURE-----

iQIzBAABCAAdFiEEEna+U/NnKOL/SmLOMoLAeC6sq98YFAMpN1vsACGkQoLAeC6sq
98YNHxAAjNzuRtk3shJaIw22T96zPNOBChIgMTR9KUo/R74b6PnwnzRyDDTWHN4
i7C4lh35XsctikBa1xinvD0WDTRgIhYbilqpODJcnVAOPw4Mol7WcvDDIM65a0ft
ELlt3PkNf3boogMBWpGnnLUZFDWrNNRiq5GRBTevQFW+AV1TF/yoTfk+/lYsx3x0
7LPmm3g3AkDp83Etq3RbqIOkgcKlE2xuM3DRoNrH1x3aLHiZ60eJh+zN0waEWyp4
91jWWUq4AnoqUESVk8El6eAiSRkD0ZWwl5iz9aHIbicyXoFD00a6cPET8hJ07rbI
08MDAuP+zWml//2JHpb+xetjdHuJSes0tfD3gL2JtuN/Sx8NLgvDgUz2E5kysgtk
pGlmgiM+xZ/GH4d5XYm0mDLsRpqHERNSCpdtj7Kfk6iq5cCo+sNgZ8Wy6bRiQQSP
bxFkeIifccuvUcpREtj9r8E9yyEldJykJfFxM9pHT0usr3SpCdRiYjJHTMBbtQXJ
haQwrYx7AAm8uedux1E44DvJ/xa9ojZDhpU0cGIn3qZgP26llQQikpGqsntfgDMp
knOrCVbvnTmLJ380obIx+OmCp1WqBAoBhPVIYGE+iVl7bGUw1iEjvySAGsd15SCR
SVt1P+Gyz05D7Q4oq7bFGoiCtqR1qAqKIN8TnyAE1CZgZdj2a2k=
=E0eY
-----END PGP SIGNATURE-----
```

Отсоединённая подпись в двоичном формате

```
smart@thinkpad$ gpg2 \
--sign \
--detach-sign \
--default-key 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \
mydocument.pdf
```

Листинг двоичного файла не имеет смысла.

Встроенная в файл подпись в текстовом формате

```
smart@thinkpad$ gpg2 \
--sign \
```

```
--default-key 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \  
--armor mydocument.pdf
```

Листинг не имеет смысла, т.к. включает объёмный исходный PDF-файл.

Встроенная в файл подпись в двоичном формате

```
smart@thinkpad$ gpg2 \  
--sign \  
--default-key 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6 \  
mydocument.pdf
```

Листинг не имеет смысла, т.к. сгенерирован бинарный файл.

### 3. Импорт открытого ключа

Возьмём файл цифровой подписи GnuPG проекта файлового шифровальщика VeraCrypt.

```
smart@thinkpad$ wget  
↪ https://www.idrix.fr/VeraCrypt/VeraCrypt_PGP_public_key.asc
```

Перед импортом, можно удостовериться, что цифровая подпись содержит правильный ID.

```
smart@thinkpad$ gpg --show-keys VeraCrypt_PGP_public_key.asc  
pub   rsa4096 2018-09-11 [SC]  
      5069A233D55A0EEB174A5FC3821ACD02680D16DE  
uid           VeraCrypt Team (2018 - Supersedes Key  
↪ ID=0x54DDD393) <veracrypt@idrix.fr>  
sub   rsa4096 2018-09-11 [E]  
sub   rsa4096 2018-09-11 [A]
```

После этого можно импортировать файл

```
smart@thinkpad$ gpg --import VeraCrypt_PGP_public_key.asc  
gpg: key 821ACD02680D16DE: 1 signature not checked due to a missing key
```

```
gpg: key 821ACD02680D16DE: public key "VeraCrypt Team (2018 - Supersedes
↳ Key ID=0x54DDD393) <veracrypt@idrix.fr>" imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2023-03-04
```

Для проверки подписи не обязательно устанавливать доверительные отношения с импортированным ключом.

## 4. Проверка ЭЦП

Для проверки, возьмём архив с исходными кодами и отсоединённую подпись в двоичном формате

```
smart@thinkpad$ wget https://launchpad.net/veracrypt/trunk/1.25.9/+download/VeraCrypt_1.25.9_Source.tar.bz2
smart@thinkpad$ wget https://launchpad.net/veracrypt/trunk/1.25.9/+download/VeraCrypt_1.25.9_Source.tar.bz2.sig
```

Теперь всё готово для проверки.

```
smart@thinkpad$ gpg --verify VeraCrypt_1.25.9_Source.tar.bz2.sig
↳ VeraCrypt_1.25.9_Source.tar.bz2
gpg: Signature made Sun 20 Feb 2022 04:18:32 PM MSK
gpg: using RSA key
↳ 5069A233D55A0EEB174A5FC3821ACD02680D16DE
gpg: Good signature from "VeraCrypt Team (2018 - Supersedes Key
↳ ID=0x54DDD393) <veracrypt@idrix.fr>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the
↳ owner.
Primary key fingerprint: 5069 A233 D55A 0EEB 174A 5FC3 821A CD02 680D
↳ 16DE
```

Ожидаемый результат – Good signature.

Остальные предупреждения, связанные с отсутствием доверия, можно игнорировать.

## 5. Экспорт/импорт на ключевые сервера

Экспорт своего ключа

```
smart@thinkpad$ gpg --keyserver hkp://keyserver.ubuntu.com --send-key
↪ 9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
gpg: sending key A0B01E0BAB2AF7C6 to hkp://keyserver.ubuntu.com
```

Теперь воспользуемся поиском, для обнаружения этого ключа на удалённом сервере

```
smart@thinkpad$ gpg --keyserver hkp://keyserver.ubuntu.com --search-keys
↪ martynov.sa@edu.spbstu.ru
gpg: data source: http://162.213.33.9:11371
(1)          Semen Martynov (Test digital sirnature)
↪ <martynov.sa@edu.spbstu.ru>
          4096 bit RSA key A0B01E0BAB2AF7C6, created: 2023-02-11
Keys 1-1 of 1 for "martynov.sa@edu.spbstu.ru". Enter number(s), N)ext,
↪ or Q)uit >

smart@thinkpad$ gpg --keyserver hkp://keyserver.ubuntu.com --list-sigs
↪ martynov.sa@edu.spbstu.ru
pub  rsa4096 2023-02-11 [SC] [expires: 2023-03-04]
      9DAF94FCD9CA38BFD298BD0CA0B01E0BAB2AF7C6
uid          [ultimate] Semen Martynov (Test digital sirnature)
↪ <martynov.sa@edu.spbstu.ru>
sig 3          A0B01E0BAB2AF7C6 2023-02-11  Semen Martynov (Test digital
↪ sirnature) <martynov.sa@edu.spbstu.ru>
sub  rsa4096 2023-02-11 [E] [expires: 2023-03-04]
sig          A0B01E0BAB2AF7C6 2023-02-11  Semen Martynov (Test digital
↪ sirnature) <martynov.sa@edu.spbstu.ru>
```

Так же легко можно импортировать ключи с удалённого сервера

```
smart@thinkpad$ gpg --keyserver pgp.mit.edu --recv-keys DAD95197
gpg: key FBF1FC87DAD95197: public key "Ashish Aniyan
↳ <aaniyan@verimatrix.com>" imported
gpg: Total number processed: 1
gpg:             imported: 1
```

Для интеграции GPG заработал в почтовом клиенте Mutt, нужно в конфигурационный файл `/.muttrc` дописать

```
set crypt_use_gpgme=yes
set crypt_autosign=yes
set crypt_replyencrypt=yes
```

## Выводы

В данной работе мы сравнили различные виды электронных цифровых подписей:

- Отсоединённую ЭЦП в текстовом формате
- Отсоединённую ЭЦП в бинарном формате
- Присоединённую ЭЦП в текстовом формате
- Присоединённую ЭЦП в бинарном формате

Работа с отсоединённой подписью удобнее. В цифровом виде подпись занимает меньше места.

В практическом плане, использование GNU Privacy Guard позволяет верифицировать не только документы, но и цифровые пакеты при распространении через Интернет.



# Лабораторная работа 5. Анализатор сетевого трафика Wireshark

**Цель работы:** Знакомство с возможностями анализатора сетевого трафика Wireshark.

## 1. Установка

Для использования wireshark в графическом режиме, потребуется пакет wireshark-qt.

```
smart@thinkpad$ sudo pacman -S wireshark-qt
resolving dependencies...
looking for conflicting packages...

Packages (10) bcg729-1.1.1-1  c-ares-1.19.0-1  cdparanoia-10.2-8
               gst-plugins-base-1.22.0-3  libmaxminddb-1.7.1-1
               qt5-multimedia-5.15.8+kde+r2-1  sbc-2.0-1  spandsp-0.0.6-4
               wireshark-cli-4.0.3-1  wireshark-qt-4.0.3-1

Total Download Size:    28.58 MiB
Total Installed Size:  138.16 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
spandsp-0.0.6-4-...  424.1 KiB  1820 KiB/s 00:00
↪ [#####] 100%
qt5-multimedia-5...  763.0 KiB  2.79 MiB/s 00:00
↪ [#####] 100%
c-ares-1.19.0-1-...  205.7 KiB  5.02 MiB/s 00:00
↪ [#####] 100%
```

```

sbc-2.0-1-x86_64      47.9 KiB  1197 KiB/s 00:00
↳ [#####] 100%
bcg729-1.1.1-1-x...   37.6 KiB   939 KiB/s 00:00
↳ [#####] 100%
libmaxminddb-1.7...   23.9 KiB  1193 KiB/s 00:00
↳ [#####] 100%
gst-plugins-base...   316.3 KiB   703 KiB/s 00:00
↳ [#####] 100%
wireshark-qt-4.0...   4.1 MiB   2.56 MiB/s 00:02
↳ [#####] 100%
wireshark-cli-4.0.3-1-x86_64
↳
22.7 MiB   3.37 MiB/s 00:07
↳ [#####]
↳ [#####]
↳ 100%
Total (9/9)
↳
28.6 MiB   4.21 MiB/s 00:07
↳ [#####]
↳ [#####]
↳ 100%
(10/10) checking keys in keyring
↳
↳ [#####]
↳ [#####]
↳ 100%
(10/10) checking package integrity
↳
↳ [#####]
↳ [#####]
↳ 100%
(10/10) loading package files
↳
↳ [#####]
↳ [#####]
↳ 100%

```

```

(10/10) checking for file conflicts
↳
↳ [#####]
↳ #####]
↳ 100%
(10/10) checking available disk space
↳
↳ [#####]
↳ #####]
↳ 100%
:: Processing package changes...
( 1/10) installing cdparanoia
↳
↳ [#####]
↳ #####]
↳ 100%
( 2/10) installing gst-plugins-base
↳
↳ [#####]
↳ #####]
↳ 100%
( 3/10) installing qt5-multimedia
↳
↳ [#####]
↳ #####]
↳ 100%
Optional dependencies for qt5-multimedia
    qt5-declarative: QML bindings [installed]
    gst-plugins-good: camera support, additional plugins
    gst-plugins-bad: camera support, additional plugins
    gst-plugins-ugly: additional plugins
    gst-libav: ffmpeg plugin
( 4/10) installing c-ares
↳
↳ [#####]
↳ #####]
↳ 100%

```

```

( 5/10) installing libmaxminddb
↳
↳ [#####]
↳ #####]
↳ 100%
Optional dependencies for libmaxminddb
    geoip2-database: IP geolocation databases
( 6/10) installing spandsp
↳
↳ [#####]
↳ #####]
↳ 100%
( 7/10) installing sbc
↳
↳ [#####]
↳ #####]
↳ 100%
( 8/10) installing bcg729
↳
↳ [#####]
↳ #####]
↳ 100%
( 9/10) installing wireshark-cli
↳
↳ [#####]
↳ #####]
↳ 100%
NOTE: To run wireshark as normal user you have to add yourself into
↳ wireshark group
(10/10) installing wireshark-qt
↳
↳ [#####]
↳ #####]
↳ 100%
:: Running post-transaction hooks...
(1/5) Creating system user accounts...
Creating group 'wireshark' with GID 150.

```

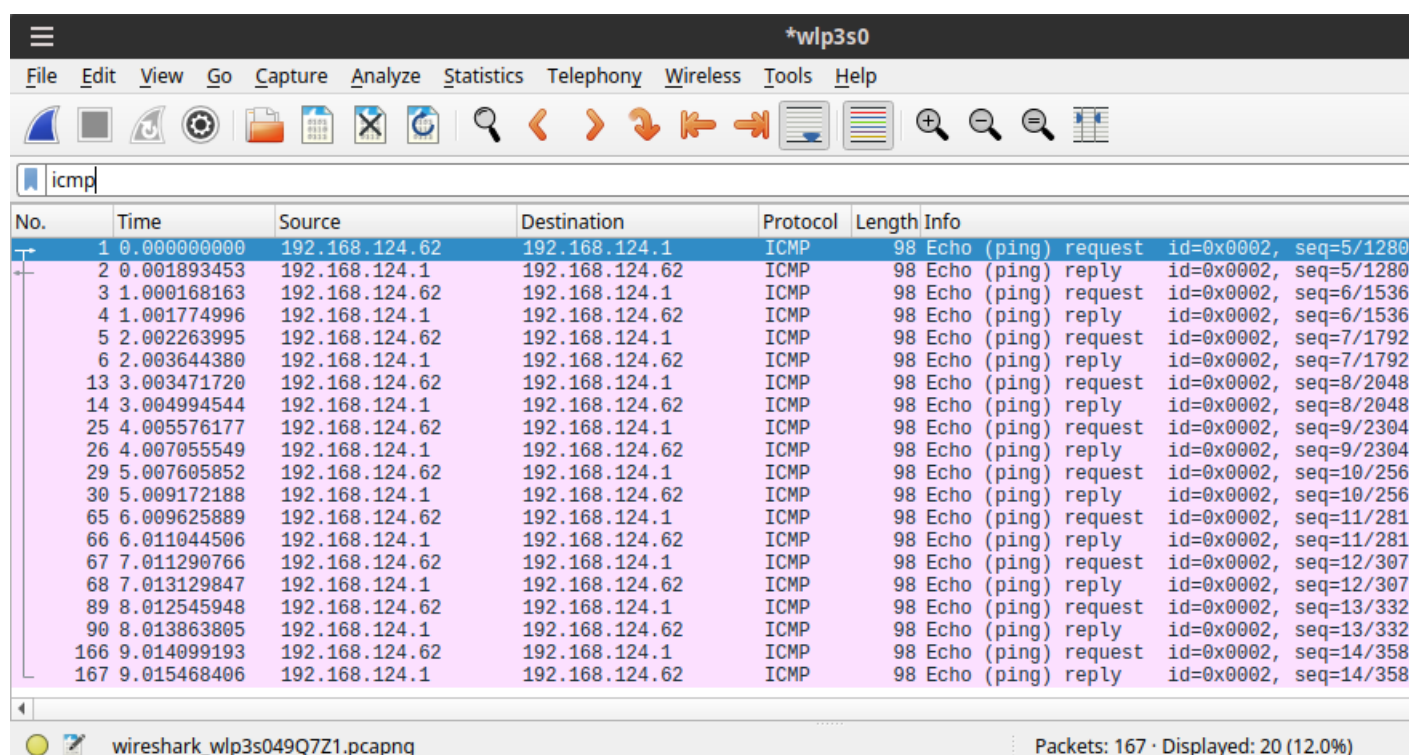
```
(2/5) Arming ConditionNeedsUpdate...
(3/5) Updating the MIME type database...
(4/5) Updating icon theme caches...
(5/5) Updating the desktop file MIME type cache...
```

## 2. Анализ ICMP трафика

Генерация трафика: Для использования `wireshark` в графическом режиме, потребуется пакет `wireshark-qt`.

```
smart@thinkpad$ ping 192.168.124.1
```

Фильтрация ICMP пакетов в `wireshark`



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=5/1280
2	0.001893453	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=5/1280
3	1.000168163	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=6/1536
4	1.001774996	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=6/1536
5	2.002263995	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=7/1792
6	2.003644380	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=7/1792
13	3.003471720	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=8/2048
14	3.004994544	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=8/2048
25	4.005576177	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=9/2304
26	4.007055549	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=9/2304
29	5.007605852	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=10/256
30	5.009172188	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=10/256
65	6.009625889	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=11/281
66	6.011044506	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=11/281
67	7.011290766	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=12/307
68	7.013129847	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=12/307
89	8.012545948	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=13/332
90	8.013863805	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=13/332
166	9.014099193	192.168.124.62	192.168.124.1	ICMP	98	Echo (ping) request id=0x0002, seq=14/358
167	9.015468406	192.168.124.1	192.168.124.62	ICMP	98	Echo (ping) reply id=0x0002, seq=14/358

Рассмотрим ICMP пакет, полученный в качестве эхо-ответа от удалённого хоста.

### Канальный уровень (data link): Ethernet фрейм

Первые 6 байт – MAC-адрес получателя пакета (14:5a:fc:0d:56:2d).

0000	14 5a fc 0d 56 2d	34 ce 00 37 d9 03 08 00 45 00
0010	00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8	
0020	7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00	
0030	00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15	
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25	
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	
0060	36 37	

Следующие 6 байт – MAC-адрес отправителя пакета (34:ce:00:37:d9:03).

0000	14 5a fc 0d 56 2d	34 ce 00 37 d9 03	08 00 45 00
0010	00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8		
0020	7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00		
0030	00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15		
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25		
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35		
0060	36 37		

Тип пакетов – IPv4 (0x0800). Полный список типов в удобном виде можно посмотреть в Wikipedia:

<https://en.wikipedia.org/wiki/EtherType>

0000	14 5a fc 0d 56 2d 34 ce 00 37 d9 03	08 00	45 00
0010	00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8		
0020	7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00		
0030	00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15		
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25		
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35		
0060	36 37		

## Сетевой уровень (network): IP пакет

Первые 4 бита указывают на версию IP. Код 0100 указывает на 4-ю версию. Структуру IPv4 пакета можно посмотреть в Wikipedia:

[https://en.wikipedia.org/wiki/Internet\\_Protocol\\_version\\_4](https://en.wikipedia.org/wiki/Internet_Protocol_version_4)

Следующие 4 бита передают длину хедера пакета. Код 0101 соответствует длине в 20 байт.

Wireshark показывает байт целиком.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Дальше идут сервисные поля DSCP (разделения трафика на классы обслуживания) и ECN (предупреждение о перегрузке сети без потери пакетов), в суть которых мы не будем углубляться, т.к. они всё равно не используются в данном случае.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Следующие 16 бит это полный размер пакета в байтах, включая заголовок и данные. По RFC, он может быть от 20 до 65535 байт. А нашем случае – 84.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Далее идёт уникальный идентификатор, используемый для идентификации фрагментов пакета, если он был фрагментирован.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Следующие три бита – поле флагов. Биты, от старшего к младшему, означают:

- 0: Зарезервирован, должен быть равен 0
- 1: Не фрагментировать
- 2: У пакета ещё есть фрагменты

И после этого ещё 13 бит для указания смещение поля данных текущего фрагмента относительно начала поля данных первого фрагментированного пакета в блоках по 8 байт.

```
0000| 14 5a fc 0d 56 2d 34 ce 00 37 d9 03 08 00 45 00
0010| 00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8
0020| 7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00
0030| 00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15
0040| 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050| 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060| 36 37
```

Время жизни пакета (Time to Live) задаёт максимальное количество маршрутизаторов (хопов) на пути следования пакета. Максимальное значение TTL=255, но чаще встречается TTL=64.

```
0000| 14 5a fc 0d 56 2d 34 ce 00 37 d9 03 08 00 45 00
0010| 00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8
0020| 7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00
0030| 00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15
0040| 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050| 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060| 36 37
```

Следующее поле показывает, данные какого протокола IP содержит пакет (к примеру, это может быть TCP). Список кодов можно посмотреть на сайте IANA:

<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml>

Код 01 соответствует ICMP.

```
0000| 14 5a fc 0d 56 2d 34 ce 00 37 d9 03 08 00 45 00
0010| 00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8
0020| 7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00
0030| 00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15
0040| 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050| 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060| 36 37
```

Контрольная сумма заголовка занимает следующие 2-байта и используемая для проверки



#### целостности заголовка

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Далее 32-битный адрес отправителя пакета (Может не совпадать с настоящим адресом отправителя из-за трансляции адресов – NAT)

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

И 32-битный адрес получателя пакета (учитывая, что это ICMP-ответ, получателем будет хост, на котором запущен ping).

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

## Сетевой уровень (network): ICMP пакет

Первый байт указывает на тип пакета.

Код 0 соответствует ICMP-ответу. Список кодов и заголовков можно посмотреть в Wikipedia:  
[https://en.wikipedia.org/wiki/Internet\\_Control\\_Message\\_Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol)

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Следующий байт – код, который зависит от типа пакета. Для ICMP-ответа возможно только 0-е значение.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Ещё два байта нужны для передачи контрольной суммы.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Следующие 4 байта так же зависят от типа и кода.

0000	14	5a	fc	0d	56	2d	34	ce	00	37	d9	03	08	00	45	00
0010	00	54	20	30	00	00	40	01	e0	e8	c0	a8	7c	01	c0	a8
0020	7c	3e	00	00	d3	bd	00	02	00	0e	7a	c1	e8	63	00	00
0030	00	00	08	3a	02	00	00	00	00	00	10	11	12	13	14	15
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24	25
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35
0060	36	37														

Далее идёт временная метка, для расчёта задержки при доставке сообщения

```

0000| 14 5a fc 0d 56 2d 34 ce 00 37 d9 03 08 00 45 00
0010| 00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8
0020| 7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00
0030| 00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15
0040| 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050| 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060| 36 37

```

Оставшиеся 48 байт присылают в содержат тело запроса, полученное ICMP-сервером (по этой причине схема и получила название "эхо").

```

0000| 14 5a fc 0d 56 2d 34 ce 00 37 d9 03 08 00 45 00
0010| 00 54 20 30 00 00 40 01 e0 e8 c0 a8 7c 01 c0 a8
0020| 7c 3e 00 00 d3 bd 00 02 00 0e 7a c1 e8 63 00 00
0030| 00 00 08 3a 02 00 00 00 00 00 10 11 12 13 14 15
0040| 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050| 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060| 36 37

```

Значения для полей при различных режимах работы ICMP приведены в Wikipedia:  
[https://en.wikipedia.org/wiki/Internet\\_Control\\_Message\\_Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol)

### 3. Анализ ARP трафика

Протокол разрешения адресов (Address Resolution Protocol, ARP) используется в компьютерных сетях для сопоставления IP-адресов и MAC-адресов в рамках одного широковещательного домена. Механизм сопоставления называют "ARP-таблицами" и этот механизм подвержен различными атакам (типа переполнения размерности) для встраивания в цепочку передачи (Man-in-the-middle).

Отображение ARP-таблицы на беспроводном интерфейсе

```

smart@thinkpad$ ip neigh show dev wlp3s0
192.168.124.1 lladdr 34:ce:00:37:d9:03 REACHABLE
fe80::36ce:ff:fe37:d903 lladdr 34:ce:00:37:d9:03 router REACHABLE

```

Фильтрация ARP пакетов в wireshark

*wlp3s0						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
arp						
No.	Time	Source	Destination	Protocol	Length	Info
1262	25.372052497	XIAOMIE1_37:d9:03	Broadcast	ARP	42	Who has 192.168.124.85? Tell 192.168.124.1
3953	44.418545724	XIAOMIE1_37:d9:03	Broadcast	ARP	42	Who has 192.168.124.62? Tell 192.168.124.1
3954	44.418572505	LiteonTe_0d:56:2d	XIAOMIE1_37:d9:03	ARP	42	192.168.124.62 is at 14:5a:fc:0d:56:2d
3955	44.421172264	XIAOMIE1_37:d9:03	LiteonTe_0d:56:2d	ARP	42	Who has 192.168.124.62? Tell 192.168.124.1
3956	44.421192532	LiteonTe_0d:56:2d	XIAOMIE1_37:d9:03	ARP	42	192.168.124.62 is at 14:5a:fc:0d:56:2d
5952	76.367643953	XIAOMIE1_37:d9:03	Broadcast	ARP	42	Who has 192.168.124.85? Tell 192.168.124.1
6033	88.402289336	XIAOMIE1_37:d9:03	LiteonTe_0d:56:2d	ARP	42	Who has 192.168.124.62? Tell 192.168.124.1
6034	88.402316858	LiteonTe_0d:56:2d	XIAOMIE1_37:d9:03	ARP	42	192.168.124.62 is at 14:5a:fc:0d:56:2d
8467	138.579281696	XIAOMIE1_37:d9:03	LiteonTe_0d:56:2d	ARP	42	Who has 192.168.124.62? Tell 192.168.124.1
8468	138.579308517	LiteonTe_0d:56:2d	XIAOMIE1_37:d9:03	ARP	42	192.168.124.62 is at 14:5a:fc:0d:56:2d
8469	138.587063420	XIAOMIE1_37:d9:03	LiteonTe_0d:56:2d	ARP	42	Who has 192.168.124.62? Tell 192.168.124.1
8470	138.587090502	LiteonTe_0d:56:2d	XIAOMIE1_37:d9:03	ARP	42	192.168.124.62 is at 14:5a:fc:0d:56:2d

## Канальный уровень (data link): Ethernet фрейм

MAC-адрес получателя ff:ff:ff:ff:ff:ff. Широковещательное (Broadcast) сообщение.

```
0000| ff ff ff ff ff ff 34 ce 00 37 d9 03 08 06 00 01
0010| 08 00 06 04 00 01 34 ce 00 37 d9 03 c0 a8 7c 01
0020| 00 00 00 00 00 00 c0 a8 7c 55
```

MAC-адрес отправителя 34:ce:00:37:d9:03

```
0000| ff ff ff ff ff ff 34 ce 00 37 d9 03 08 06 00 01
0010| 08 00 06 04 00 01 34 ce 00 37 d9 03 c0 a8 7c 01
0020| 00 00 00 00 00 00 c0 a8 7c 55
```

Тип сообщения – ARP (0x0806). Справочник в Wikipedia:

<https://en.wikipedia.org/wiki/EtherType>

```
0000| ff ff ff ff ff ff 34 ce 00 37 d9 03 08 06 00 01
0010| 08 00 06 04 00 01 34 ce 00 37 d9 03 c0 a8 7c 01
0020| 00 00 00 00 00 00 c0 a8 7c 55
```

## Сетевой уровень (network): ARP пакет

Структура пакетов и значения флагов описаны в Wikipedia:

<https://ru.wikipedia.org/wiki/ARP>

Первые два байта – тип канального протокола. Ethernet имеет номер 0x0001.

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Ещё два байта для кода сетевого протокола. Для IPv4 код 0x0800.

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Длина физического адреса в байтах. Адреса Ethernet имеют длину 6 байт (0x06)

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Длина логического адреса в байтах. IPv4 адреса имеют длину 4 байта (0x04).

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Код операции отправителя: 0x0001 в случае запроса и 0x0002 в случае ответа.

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Физический адрес отправителя (SHA)

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Логический адрес отправителя (SPA)

0000	ff	ff	ff	ff	ff	ff	34	ce	00	37	d9	03	08	06	00 01
0010	08	00	06	04	00	01	34	ce	00	37	d9	03	c0	a8	7c 01
0020	00	00	00	00	00	00	c0	a8	7c	55					

Физический адрес получателя (THA). Не требуется при запросе.

```

0000| ff ff ff ff ff ff 34 ce 00 37 d9 03 08 06 00 01
0010| 08 00 06 04 00 01 34 ce 00 37 d9 03 c0 a8 7c 01
0020| 00 00 00 00 00 00 c0 a8 7c 55

```

Логический адрес получателя(ТРА).

```

0000| ff ff ff ff ff ff 34 ce 00 37 d9 03 08 06 00 01
0010| 08 00 06 04 00 01 34 ce 00 37 d9 03 c0 a8 7c 01
0020| 00 00 00 00 00 00 c0 a8 7c 55

```

### 3. Анализ FTP трафика

Для работы с FTP, используем проект `vsftpd` (very secure FTP daemon).

Так же создадим виртуальный сервер с публичным IP адресом и присвоим ему доменное имя (это понадобится в будущем для получения TLS-сертификатов).

#### Демонстрация уязвимости

Осуществим запуск `vsftd` в docker-контейнере. Имя пользователя `one`, пароль `1234`.

```

ubuntu@temp$ docker run -d \
    -p 21:21 \
    -p 21000-21010:21000-21010 \
    -e USERS="one|1234" \
    -e ADDRESS=ftp.deeprosoft.com \
    delfer/alpine-ftp-server
Unable to find image 'delfer/alpine-ftp-server:latest' locally
latest: Pulling from delfer/alpine-ftp-server
df9b9388f04a: Pull complete
dff9a38fdd78: Pull complete
961dffff6741: Pull complete
99d694c3fc07: Pull complete
2dbb22f0414d: Pull complete
Digest: sha256:b030e5ee82965fb7d7135f8dbffd77c9b753ef1b60d22d1f4eb92d4d3
↪ 7f71c13
Status: Downloaded newer image for delfer/alpine-ftp-server:latest
a188e56c28dca2e19e517960b2ae526c306147d24fa364471368f79debd75e08

```

FTP является устаревшим протоколом, и его поддержка давно удалена из браузеров. По этой причине необходимо явным образом установить FTP-клиент.

```
smart@thinkpad$ sudo pacman -S filezilla
[sudo] password for smart:
resolving dependencies...
looking for conflicting packages...

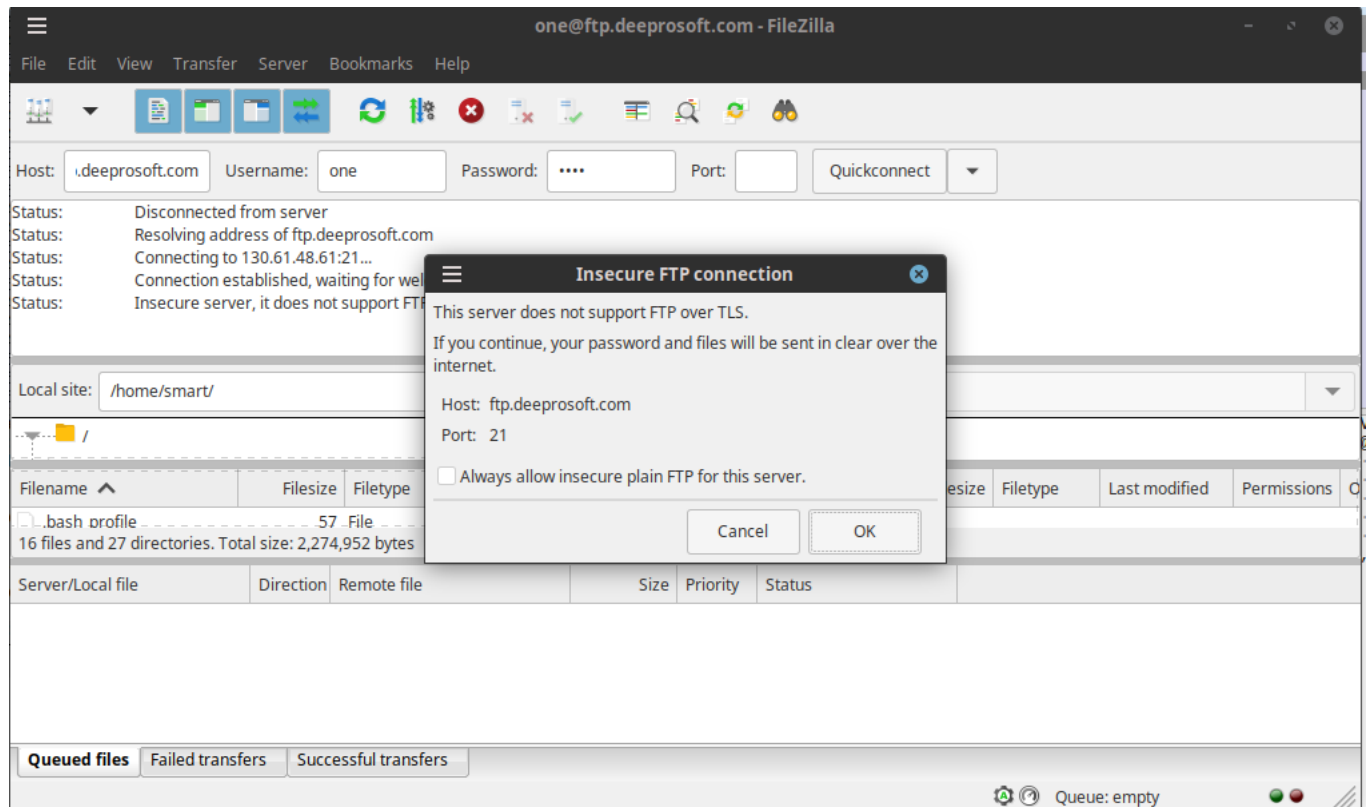
Packages (2) libfilezilla-1:0.41.0-1 filezilla-3.63.1-1

Total Download Size:    4.69 MiB
Total Installed Size:  17.54 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
libfilezilla-1:0...  405.5 KiB  1179 KiB/s 00:00
↳ [#####] 100%
filezilla-3.63.1...   4.3 MiB  4.63 MiB/s 00:01
↳ [#####] 100%
Total (2/2)           4.7 MiB  4.81 MiB/s 00:01
↳ [#####] 100%
(2/2) checking keys in keyring
↳ [#####] 100%
(2/2) checking package integrity
↳ [#####] 100%
(2/2) loading package files
↳ [#####] 100%
(2/2) checking for file conflicts
↳ [#####] 100%
(2/2) checking available disk space
↳ [#####] 100%
:: Processing package changes...
(1/2) installing libfilezilla
↳ [#####] 100%
(2/2) installing filezilla
↳ [#####] 100%
:: Running post-transaction hooks...
```

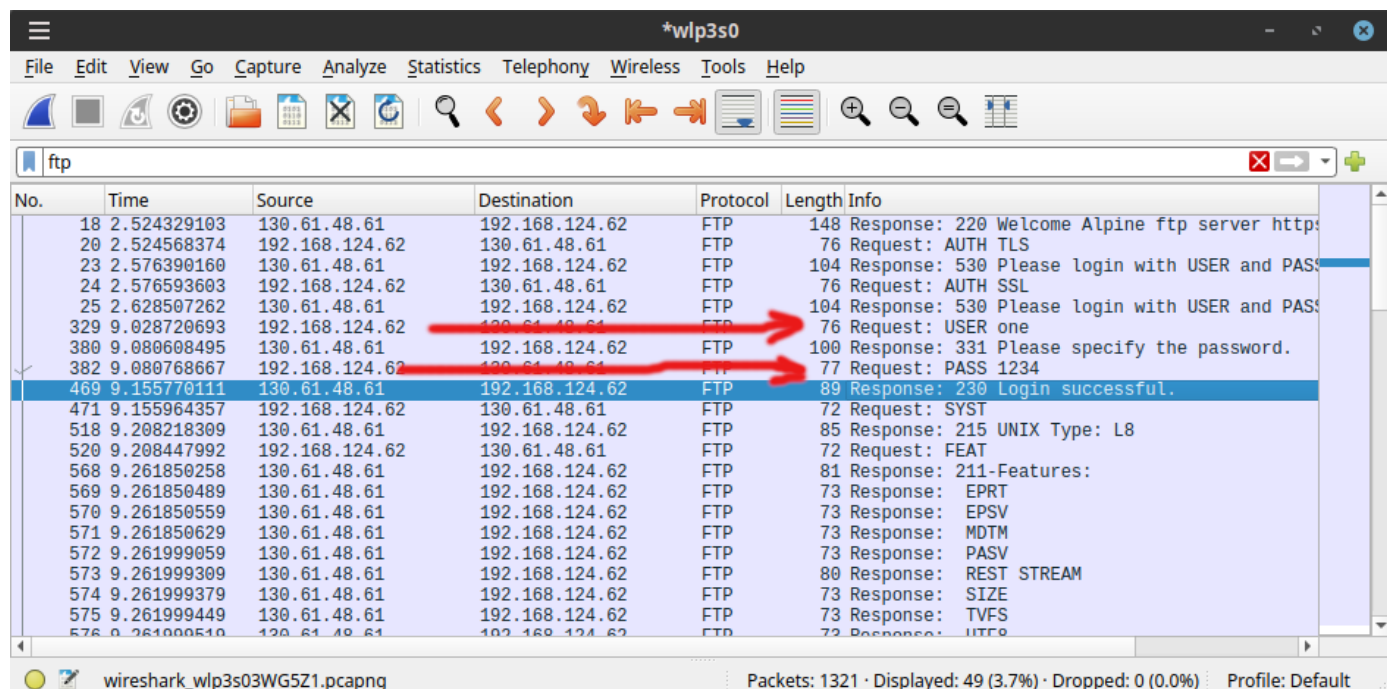
(1/3) Arming ConditionNeedsUpdate...  
(2/3) Updating icon theme caches...  
(3/3) Updating the desktop file MIME type cache...

Ещё на стадии подключения, filezilla предупреждает о небезопасности подобного подключения





При анализе трафика в wireshark, мы видим что со стороны filezilla была попытка установить соединение с использованием TLS и SSL, и только после этого имя пользователя и пароль были переданы в виде чистого текста.



No.	Time	Source	Destination	Protocol	Length	Info
18	2.524329103	130.61.48.61	192.168.124.62	FTP	148	Response: 220 Welcome Alpine ftp server https
20	2.524568374	192.168.124.62	130.61.48.61	FTP	76	Request: AUTH TLS
23	2.576390160	130.61.48.61	192.168.124.62	FTP	104	Response: 530 Please login with USER and PASS
24	2.576593603	192.168.124.62	130.61.48.61	FTP	76	Request: AUTH SSL
25	2.628507262	130.61.48.61	192.168.124.62	FTP	104	Response: 530 Please login with USER and PASS
329	9.028720693	192.168.124.62	130.61.48.61	FTP	76	Request: USER one
380	9.080608495	130.61.48.61	192.168.124.62	FTP	100	Response: 331 Please specify the password.
382	9.080768667	192.168.124.62	130.61.48.61	FTP	77	Request: PASS 1234
469	9.155770111	130.61.48.61	192.168.124.62	FTP	89	Response: 230 Login successful.
471	9.155964357	192.168.124.62	130.61.48.61	FTP	72	Request: SYST
518	9.208218309	130.61.48.61	192.168.124.62	FTP	85	Response: 215 UNIX Type: L8
520	9.208447992	192.168.124.62	130.61.48.61	FTP	72	Request: FEAT
568	9.261850258	130.61.48.61	192.168.124.62	FTP	81	Response: 211-Features:
569	9.261850489	130.61.48.61	192.168.124.62	FTP	73	Response: EPRT
570	9.261850559	130.61.48.61	192.168.124.62	FTP	73	Response: EPSV
571	9.261850629	130.61.48.61	192.168.124.62	FTP	73	Response: MDTM
572	9.261999059	130.61.48.61	192.168.124.62	FTP	73	Response: PASV
573	9.261999309	130.61.48.61	192.168.124.62	FTP	80	Response: REST STREAM
574	9.261999379	130.61.48.61	192.168.124.62	FTP	73	Response: SIZE
575	9.261999449	130.61.48.61	192.168.124.62	FTP	73	Response: TVFS
576	9.261999510	130.61.48.61	192.168.124.62	FTP	73	Response: UTF8

## Шифрование трафика

Никакие изменения настроек (смена банеров приветствия, отключение анонимного доступа) vsftd не решит проблему передачи логина и пароля в чистом виде. Необходимо организовать обеспечить шифрование канала, а для этого нужно использовать валидные TLS-сертификаты (воспользуемся службой LetsEncrypt).

```
ubuntu@temp$ docker run -it --rm \
  -p 80:80 \
  -v "/etc/letsencrypt:/etc/letsencrypt" \
  certbot/certbot certonly \
  --standalone \
  --preferred-challenges http \
  -n --agree-tos \
  --email martynov.sa@edu.spbstu.ru \
  -d ftp.deeprosoft.com
Unable to find image 'certbot/certbot:latest' locally
```

```

latest: Pulling from certbot/certbot
ca7dd9ec2225: Pull complete
9e124a36b9ab: Pull complete
42cba90def7f: Pull complete
036c0ab6a768: Pull complete
de6312618cf7: Pull complete
f2b159e8e18d: Pull complete
5c3094a661e9: Pull complete
ae4269d8cd1f: Pull complete
fc17a613a054: Pull complete
7560c853872e: Pull complete
ab060fadf2d2: Pull complete
b81696353590: Pull complete
144b4c29fbe6: Pull complete
Digest: sha256:f632e55104da84ba5b54bc858a67ac6c9b4f68790ea823515867c9931
↳ 53c3fb4
Status: Downloaded newer image for certbot/certbot:latest
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Account registered.
Requesting a certificate for ftp.deeprosoft.com

Successfully received certificate.
Certificate is saved at:
↳ /etc/letsencrypt/live/ftp.deeprosoft.com/fullchain.pem
Key is saved at:
↳ /etc/letsencrypt/live/ftp.deeprosoft.com/privkey.pem
This certificate expires on 2023-05-13.
These files will be updated when the certificate renews.

NEXT STEPS:
- The certificate will need to be renewed before it expires. Certbot can
↳ automatically renew the certificate in the background, but you may
↳ need to take steps to enable that functionality. See
↳ https://certbot.org/renewal-setup for instructions.

- - - - -
↳ - - - -

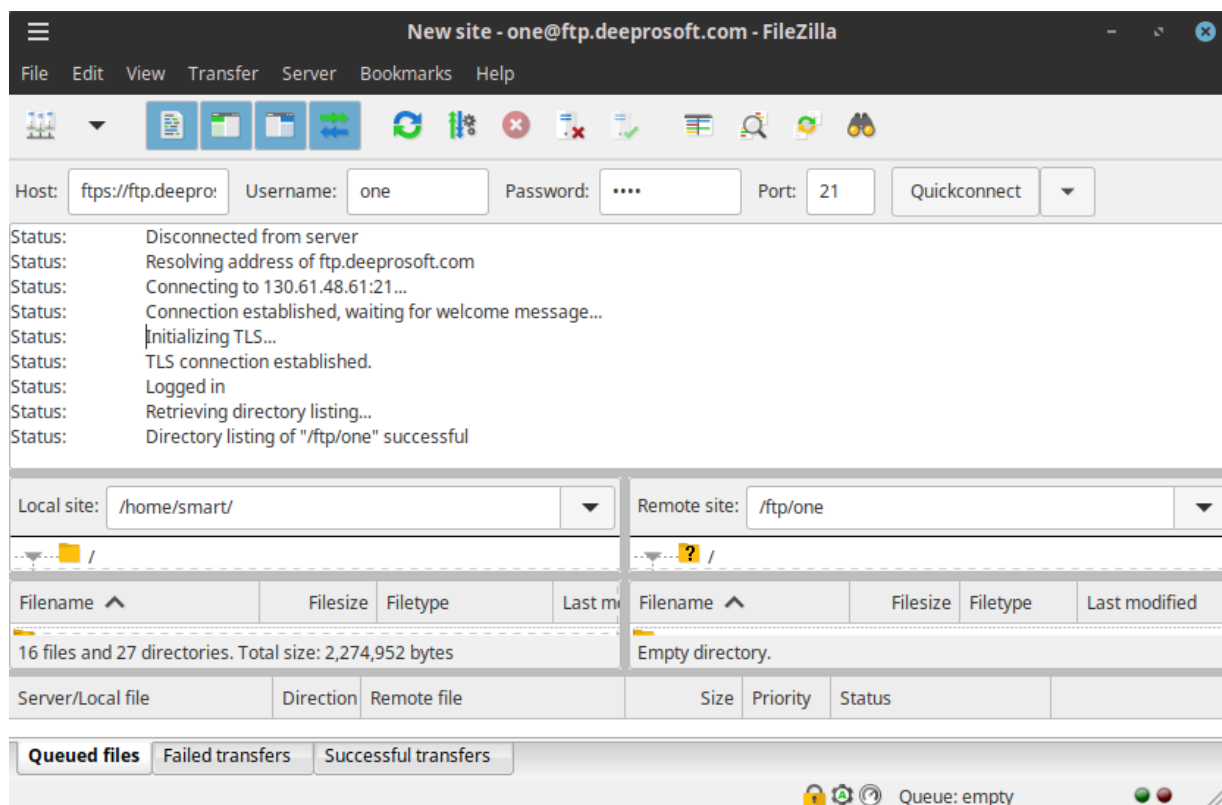
```

```
If you like Certbot, please consider supporting our work by:
* Donating to ISRG / Let's Encrypt:  https://letsencrypt.org/donate
* Donating to EFF:                    https://eff.org/donate-le
- - - - -
↪ - - - -
```

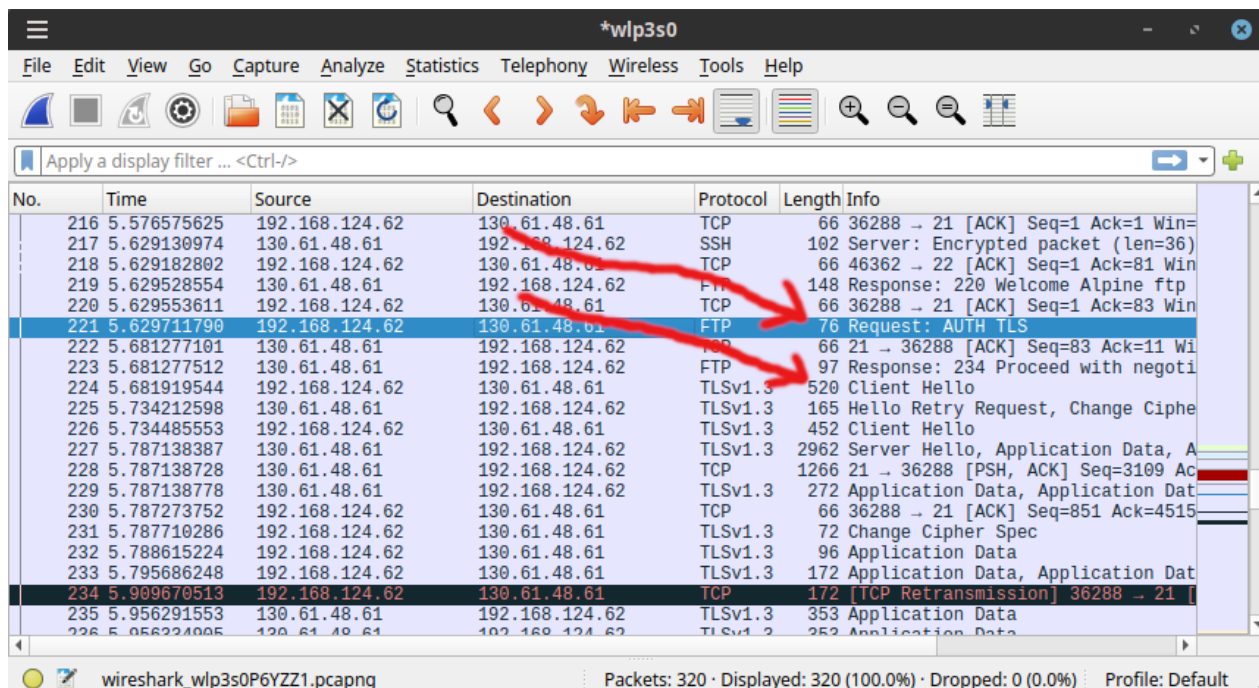
Теперь запустим `vsftd`, используя полученные TLS-сертификаты.

```
ubuntu@temp$ docker run -d \
  --name ftp \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  -p 21:21 \
  -p 21000-21010:21000-21010 \
  -e USERS="one|1234" \
  -e ADDRESS=ftp.deeprosoft.com \
  -e TLS_CERT="/etc/letsencrypt/live/ftp.deeprosoft.com/fullchain.pem"
↪ \
  -e TLS_KEY="/etc/letsencrypt/live/ftp.deeprosoft.com/privkey.pem" \
  delfer/alpine-ftp-server
02c57ccb66c3179da831927af6fe53c29a2ab59926e0901b0fb189ddbc66d06c
```

Произведём подключение по протоколу ftps.



Снифер wireshark уже не может перехватить пароль.



## 4. Сравнение защиты Telnet и SSH

Подготовим и запустим Telnet сервер в Docker окружении.

```
smart@thinkpad$ docker run -itd -p 23:23 --name=telnetServer
↳ flemingcsi/telnet-server
Unable to find image 'flemingcsi/telnet-server:latest' locally
latest: Pulling from flemingcsi/telnet-server
c3b9c0688e3b: Pull complete
e9fb5affebb0: Pull complete
0f1378f511ad: Pull complete
96a961dc7843: Pull complete
16564141bc83: Pull complete
13bab7266e87: Pull complete
Digest: sha256:d667ca1bf508945dffe1de37812bac96469041471f9a1831567cf5400
↳ 4c7e400
Status: Downloaded newer image for flemingcsi/telnet-server:latest
066114bb8b7fbb57985e5289bfedc17c5634ac8d9fb2ffaca75bac66b0e075e4
```

Уточняю, какой адрес у запущенного сервиса

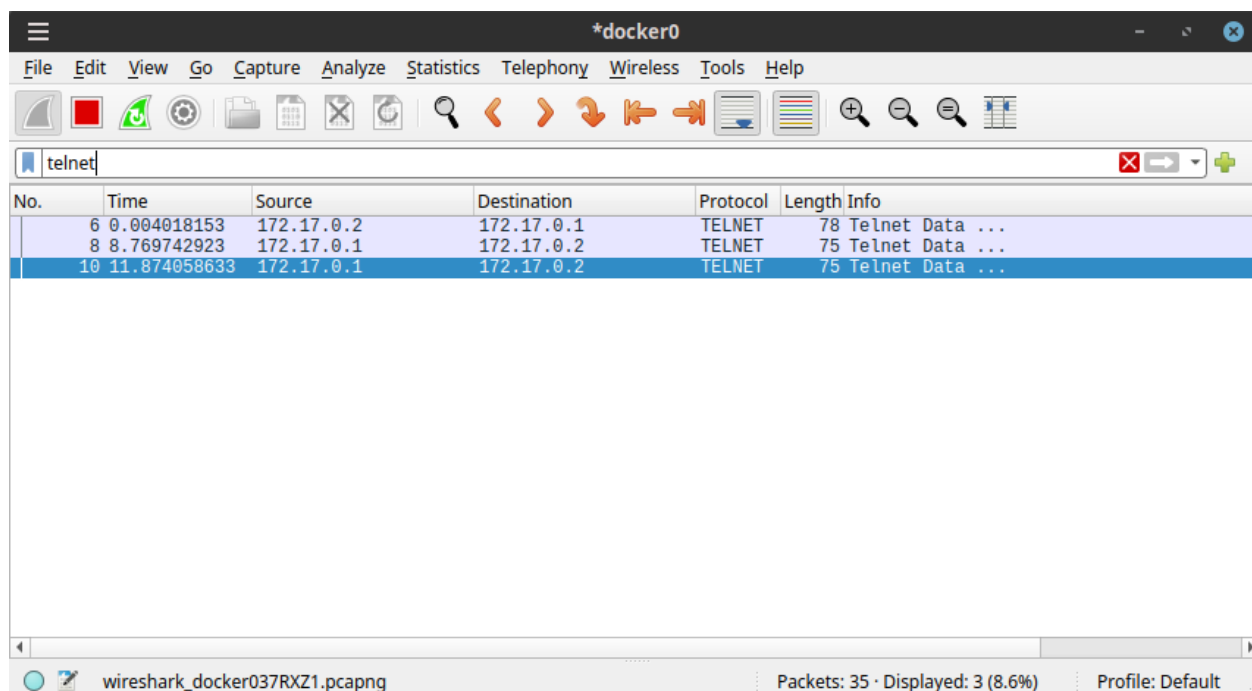
```
smart@thinkpad$ docker inspect --format={{.NetworkSettings.IPAddress}}
↳ telnetServer
172.17.0.2
```

Теперь можно воспользоваться nc в качестве telnet-клиента.

При подключении есть какие-то проблемы с кодировкой, но это не важно.

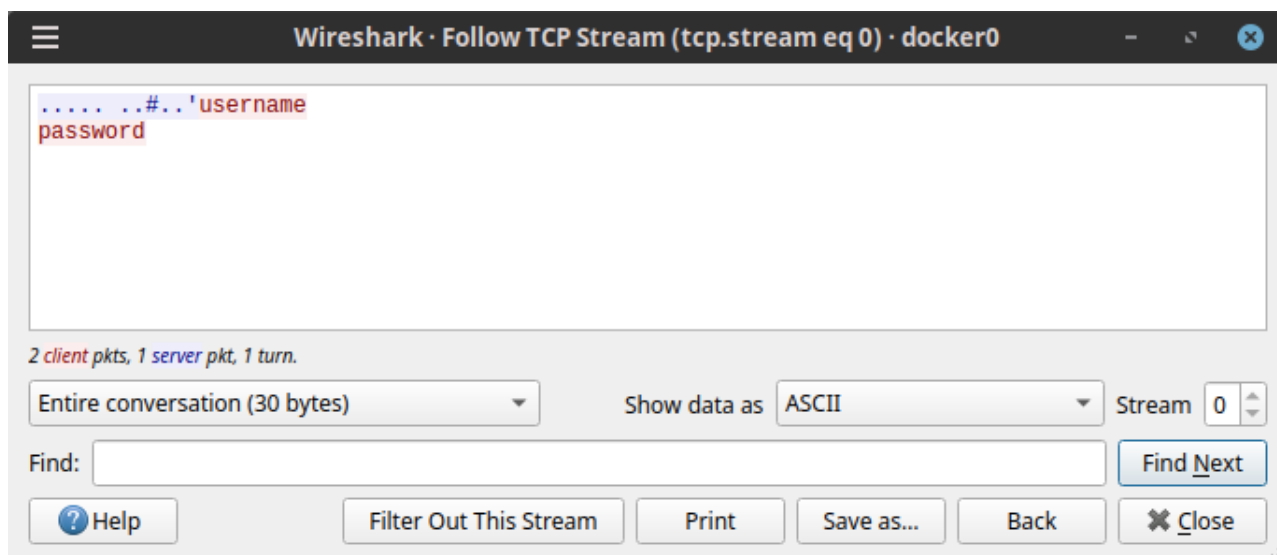
```
smart@thinkpad$ nc 127.14.0.2 23
???? ??????'username
password
```

Снифер wireshark легко перехватывает все сообщения.

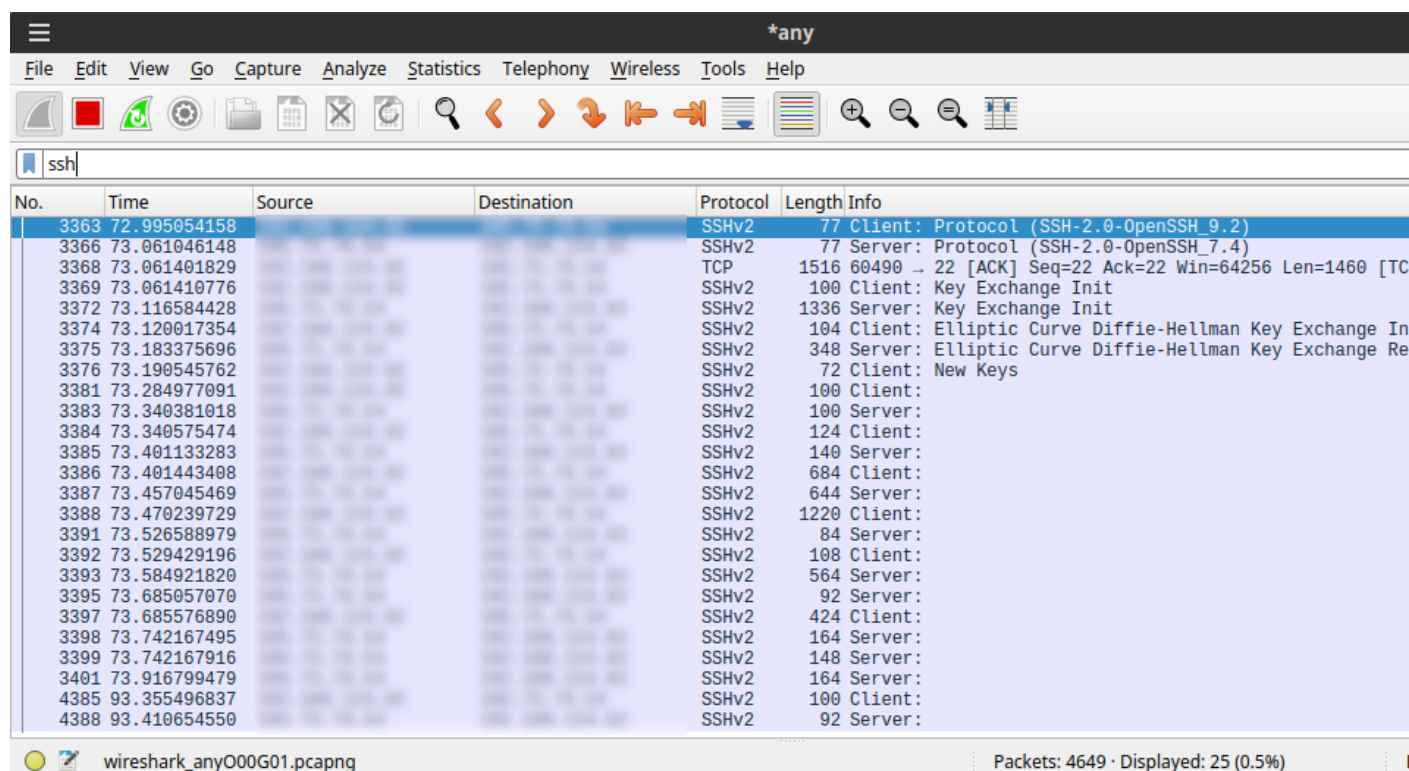


В меню можно выбрать "Follow TCP stream" для более удобного прочтения перехваченных данных.

Можно без труда получить имя пользователя и пароль, можно просматривать все данные, передаваемые в обе стороны и даже подменять их на лету.

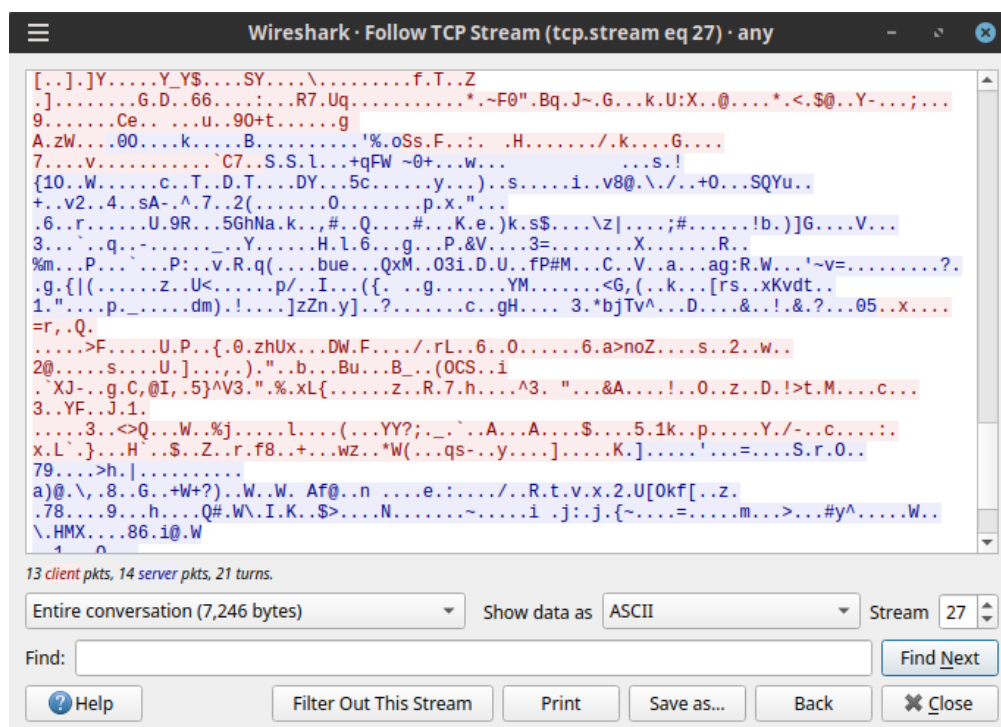


При подключении по SSH можно только увидеть IP-адреса клиента и сервера, а так же процесс согласования ключей на эллиптических кривых.



No.	Time	Source	Destination	Protocol	Length	Info
3363	72.995054158			SSHv2	77	Client: Protocol (SSH-2.0-OpenSSH_9.2)
3366	73.061046148			SSHv2	77	Server: Protocol (SSH-2.0-OpenSSH_7.4)
3368	73.061401829			TCP	1516	60490 → 22 [ACK] Seq=22 Ack=22 Win=64256 Len=1460 [TC...]
3369	73.061410776			SSHv2	100	Client: Key Exchange Init
3372	73.116584428			SSHv2	1336	Server: Key Exchange Init
3374	73.120017354			SSHv2	104	Client: Elliptic Curve Diffie-Hellman Key Exchange In...
3375	73.183375696			SSHv2	348	Server: Elliptic Curve Diffie-Hellman Key Exchange Re...
3376	73.190545762			SSHv2	72	Client: New Keys
3381	73.284977091			SSHv2	100	Client:
3383	73.340381018			SSHv2	100	Server:
3384	73.340575474			SSHv2	124	Client:
3385	73.401133283			SSHv2	140	Server:
3386	73.401443408			SSHv2	684	Client:
3387	73.457045469			SSHv2	644	Server:
3388	73.470239729			SSHv2	1220	Client:
3391	73.526588979			SSHv2	84	Server:
3392	73.529429196			SSHv2	108	Client:
3393	73.584921820			SSHv2	564	Server:
3395	73.685057070			SSHv2	92	Server:
3397	73.685576890			SSHv2	424	Client:
3398	73.742167495			SSHv2	164	Server:
3399	73.742167916			SSHv2	148	Server:
3401	73.916799479			SSHv2	164	Server:
4385	93.355496837			SSHv2	100	Client:
4388	93.410654550			SSHv2	92	Server:

Просмотреть какой-то трафик не получится, т.к. он надёжно шифруется.

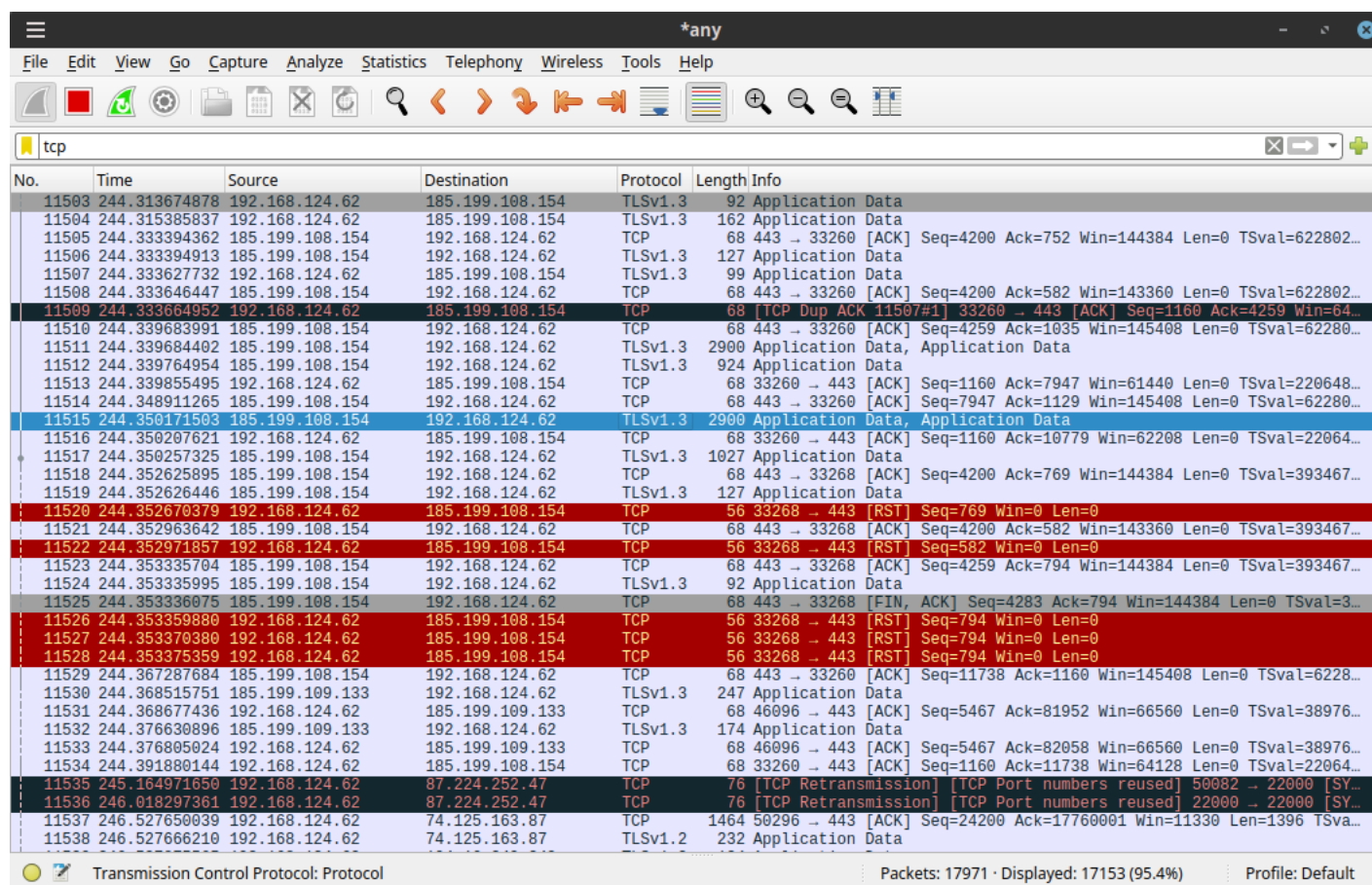




## 5. Анализ сообщений транспортного уровня: UDP-дейтаграммы и TCP-сегменты

Все различия происходят из архитектуры этих протоколов.

При фильтрации по TCP, особенно при интенсивном трафике, мы видим большое количество пакетов, связанных с исправлением различных ошибочных ситуаций. При фильтрации по UDP мы видим много пакетов, задача которых в нотификации.



No.	Time	Source	Destination	Protocol	Length	Info
11503	244.313674878	192.168.124.62	185.199.108.154	TLSv1.3	92	Application Data
11504	244.315385837	192.168.124.62	185.199.108.154	TLSv1.3	162	Application Data
11505	244.333394362	185.199.108.154	192.168.124.62	TCP	68	443 → 33260 [ACK] Seq=4200 Ack=752 Win=144384 Len=0 TSval=622802...
11506	244.333394913	185.199.108.154	192.168.124.62	TLSv1.3	127	Application Data
11507	244.333627732	192.168.124.62	185.199.108.154	TLSv1.3	99	Application Data
11508	244.333646447	185.199.108.154	192.168.124.62	TCP	68	443 → 33260 [ACK] Seq=4200 Ack=582 Win=143360 Len=0 TSval=622802...
11509	244.333664952	192.168.124.62	185.199.108.154	TCP	68	[TCP Dup ACK 11507#1] 33260 → 443 [ACK] Seq=1160 Ack=4259 Win=64...
11510	244.339683991	185.199.108.154	192.168.124.62	TCP	68	443 → 33260 [ACK] Seq=4259 Ack=1035 Win=145408 Len=0 TSval=62280...
11511	244.339684402	185.199.108.154	192.168.124.62	TLSv1.3	2900	Application Data, Application Data
11512	244.339764954	185.199.108.154	192.168.124.62	TLSv1.3	924	Application Data
11513	244.339855495	192.168.124.62	185.199.108.154	TCP	68	33260 → 443 [ACK] Seq=1160 Ack=7947 Win=61440 Len=0 TSval=220648...
11514	244.348911265	185.199.108.154	192.168.124.62	TCP	68	443 → 33260 [ACK] Seq=7947 Ack=1129 Win=145408 Len=0 TSval=62280...
11515	244.350171503	185.199.108.154	192.168.124.62	TLSv1.3	2900	Application Data, Application Data
11516	244.350207621	192.168.124.62	185.199.108.154	TCP	68	33260 → 443 [ACK] Seq=1160 Ack=10779 Win=62208 Len=0 TSval=22064...
11517	244.350257325	185.199.108.154	192.168.124.62	TLSv1.3	1027	Application Data
11518	244.352625895	185.199.108.154	192.168.124.62	TCP	68	443 → 33268 [ACK] Seq=4200 Ack=769 Win=144384 Len=0 TSval=393467...
11519	244.352626446	185.199.108.154	192.168.124.62	TLSv1.3	127	Application Data
11520	244.352670379	192.168.124.62	185.199.108.154	TCP	56	33268 → 443 [RST] Seq=769 Win=0 Len=0
11521	244.352963642	185.199.108.154	192.168.124.62	TCP	68	443 → 33268 [ACK] Seq=4200 Ack=582 Win=143360 Len=0 TSval=393467...
11522	244.352971857	192.168.124.62	185.199.108.154	TCP	56	33268 → 443 [RST] Seq=582 Win=0 Len=0
11523	244.353335704	185.199.108.154	192.168.124.62	TCP	68	443 → 33268 [ACK] Seq=4259 Ack=794 Win=144384 Len=0 TSval=393467...
11524	244.353335995	185.199.108.154	192.168.124.62	TLSv1.3	92	Application Data
11525	244.353336075	185.199.108.154	192.168.124.62	TCP	68	443 → 33268 [FIN, ACK] Seq=4283 Ack=794 Win=144384 Len=0 TSval=3...
11526	244.353359880	192.168.124.62	185.199.108.154	TCP	56	33268 → 443 [RST] Seq=794 Win=0 Len=0
11527	244.353370380	192.168.124.62	185.199.108.154	TCP	56	33268 → 443 [RST] Seq=794 Win=0 Len=0
11528	244.353375359	192.168.124.62	185.199.108.154	TCP	56	33268 → 443 [RST] Seq=794 Win=0 Len=0
11529	244.367287684	185.199.108.154	192.168.124.62	TCP	68	443 → 33260 [ACK] Seq=11738 Ack=1160 Win=145408 Len=0 TSval=6228...
11530	244.368615751	185.199.109.133	192.168.124.62	TLSv1.3	247	Application Data
11531	244.368677436	192.168.124.62	185.199.109.133	TCP	68	46096 → 443 [ACK] Seq=5467 Ack=81952 Win=66560 Len=0 TSval=38976...
11532	244.376630896	185.199.109.133	192.168.124.62	TLSv1.3	174	Application Data
11533	244.376805024	192.168.124.62	185.199.109.133	TCP	68	46096 → 443 [ACK] Seq=5467 Ack=82058 Win=66560 Len=0 TSval=38976...
11534	244.391880144	192.168.124.62	185.199.108.154	TCP	68	33260 → 443 [ACK] Seq=1160 Ack=11738 Win=64128 Len=0 TSval=22064...
11535	245.164971650	192.168.124.62	87.224.252.47	TCP	76	[TCP Retransmission] [TCP Port numbers reused] 50082 → 22000 [SY...
11536	246.018297361	192.168.124.62	87.224.252.47	TCP	76	[TCP Retransmission] [TCP Port numbers reused] 22000 → 22000 [SY...
11537	246.527650039	192.168.124.62	74.125.163.87	TCP	1464	50296 → 443 [ACK] Seq=24200 Ack=17760001 Win=11330 Len=1396 TSva...
11538	246.527666210	192.168.124.62	74.125.163.87	TLSv1.2	232	Application Data

В каждый пакет данных TCP добавляет заголовок общим объемом в 20 байт (или октетов), в котором содержатся 10 обязательных полей:

- Порт источника – порт устройства-отправителя.
- Порт назначения – порт принимающего устройства.
- Порядковый номер – Устройство, инициирующее TCP-соединение, должно выбрать случайный начальный порядковый номер, который затем увеличивается в соответствии с количеством переданных байтов.



- Номер подтверждения – Принимающее устройство увеличивает этот номер с нуля в соответствии с количеством полученных байтов.
- Сдвиг данных ТСР – Данный параметр определяет размер заголовка, чтобы система могла понять, где начинаются данные.
- Зарезервированные данные – зарезервированное поле, значение которого всегда равно нулю.
- Флаги управления – ТСР использует девять флагов для управления потоком данных в определенных ситуациях (например, при инициировании сброса сессии).
- Контрольная сумма – Отправитель генерирует контрольную сумму и передает ее в заголовке каждого пакета. Принимающее устройство может использовать контрольную сумму для проверки ошибок в полученном файле.
- Срочный указатель – это предлагаемая протоколом возможность пометить некоторые байты данных тегом «Срочно» для их пересылки и обработки вне очереди.
- Поле опции – Может использоваться для расширения протокола или его тестирования.

*any						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
[Icons]						
udp						
No.	Time	Source	Destination	Protocol	Length	Info
14225	306.966036383	192.168.124.1	239.255.255.250	SSDP	504	NOTIFY * HTTP/1.1
14226	306.966831454	192.168.124.1	239.255.255.250	SSDP	522	NOTIFY * HTTP/1.1
14227	306.967580648	192.168.124.1	239.255.255.250	SSDP	506	NOTIFY * HTTP/1.1
14228	306.968381880	192.168.124.1	239.255.255.250	SSDP	506	NOTIFY * HTTP/1.1
14229	307.165726410	192.168.124.1	239.255.255.250	SSDP	506	NOTIFY * HTTP/1.1
14230	307.166502665	192.168.124.1	239.255.255.250	SSDP	506	NOTIFY * HTTP/1.1
14231	307.167300851	192.168.124.1	239.255.255.250	SSDP	522	NOTIFY * HTTP/1.1
14232	307.168066917	192.168.124.1	239.255.255.250	SSDP	504	NOTIFY * HTTP/1.1
14233	307.168752921	192.168.124.1	239.255.255.250	SSDP	451	NOTIFY * HTTP/1.1
14234	307.169510461	192.168.124.1	239.255.255.250	SSDP	490	NOTIFY * HTTP/1.1
14235	307.170217375	192.168.124.1	239.255.255.250	SSDP	451	NOTIFY * HTTP/1.1
14236	307.170986727	192.168.124.1	239.255.255.250	SSDP	510	NOTIFY * HTTP/1.1
14237	307.171694182	192.168.124.1	239.255.255.250	SSDP	451	NOTIFY * HTTP/1.1
14238	307.172499021	192.168.124.1	239.255.255.250	SSDP	514	NOTIFY * HTTP/1.1
14239	307.173179214	192.168.124.1	239.255.255.250	SSDP	442	NOTIFY * HTTP/1.1
14240	308.737187344	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 0, PADDING...
14241	308.938032681	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 1, PADDING...
14242	308.938059171	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 2, PADDING...
14243	309.338429793	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 3, PADDING...
14244	309.338457976	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 4, PADDING...
14245	310.139561495	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 5, PADDING...
14246	310.139589738	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 6, PADDING...
14254	311.741687288	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 7, PADDING...
14255	311.741704100	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=0e7bdde5d42a957ef8, SCID=aa93f121, PKN: 8, PADDING...
15196	332.466868886	192.168.124.62	192.168.124.1	DNS	94	Standard query 0xc141 A mobile.events.data.microsoft.com
15197	332.466885658	192.168.124.62	192.168.124.1	DNS	94	Standard query 0x62ba AAAA mobile.events.data.microsoft.com
15198	332.472522986	192.168.124.1	192.168.124.62	DNS	539	Standard query response 0xc141 A mobile.events.data.microsoft.co...
15199	332.473451188	192.168.124.1	192.168.124.62	DNS	259	Standard query response 0x62ba AAAA mobile.events.data.microsoft...
15228	333.739637470	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 0, P...
15229	333.940716608	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 1, P...
15230	333.940743278	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 2, P...
15231	334.341808431	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 3, P...
15232	334.341845221	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 4, P...
15233	335.142803815	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 5, P...
15234	335.142820206	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 6, P...
15235	336.746174328	192.168.124.62	87.224.252.47	QUIC	1296	Initial, DCID=83335d773d6b11c99d1b511d, SCID=a05c32bf, PKN: 7, P...



User Datagram Protocol: Protocol

Packets: 20152 · Displayed: 733 (3.6%)

Profile: Default

Заголовки UDP значительно проще.

- Порт отправителя – номер порта отправителя
- Порт получателя – содержит порт получателя
- Длина датаграммы – Поле, задающее длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка – 8 байт. Теоретически, максимальный размер поля – 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные).

## Выводы

В реальной жизни достаточно тяжело встретиться с такими устаревшими протоколами как FTP или Telnet, но полезно знать об их уязвимостях. В процессе работы использовался сетевой сниффер `wireshark`. На практике инженеры обычно используют `tcpdump`, но `wireshark` позволяет легко визуализировать трафик и разложить пакет по битам.

Так же было проведено сравнение между TCP и UDP. Ключевым различием между TCP и UDP является скорость, поскольку TCP сравнительно сложнее UDP. В целом, UDP является быстрым, простым и эффективным протоколом, однако повторная передача потерянных пакетов данных возможна только в TCP.

# Лабораторная работа 6. Аудит защищенности сети сканером Nmap

**Цель работы:** Знакомство с возможностями сетевого сканера Nmap.

## 1. Установка

Для использования сканера, необходимо произвести установку соответствующего пакета.

```
smart@thinkpad$ sudo pacman -S nmap
[sudo] password for smart:
resolving dependencies...
looking for conflicting packages...

Packages (2) lua53-5.3.6-1  nmap-7.93-1

Total Download Size:    5.78 MiB
Total Installed Size:  25.29 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
lua53-5.3.6-1-x86_64                253.2 KiB   882 KiB/s  00:00
  ↳ [#####] 100%
nmap-7.93-1-x86_64                  5.5 MiB   4.33 MiB/s  00:01
  ↳ [#####] 100%
Total (2/2)                          5.8 MiB   4.41 MiB/s  00:01
  ↳ [#####] 100%
(2/2) checking keys in keyring
  ↳ [#####] 100%
```

```
(2/2) checking package integrity
↳ [#####] 100%
(2/2) loading package files
↳ [#####] 100%
(2/2) checking for file conflicts
↳ [#####] 100%
(2/2) checking available disk space
↳ [#####] 100%
:: Processing package changes...
(1/2) installing lua53
↳ [#####] 100%
(2/2) installing nmap
↳ [#####] 100%
:: Running post-transaction hooks...
(1/1) Arming ConditionNeedsUpdate...
```

## 2. Обнаружение хостов в сети

Обнаружение хостов может стать первым шагом при исследовании сети.

### 2.1. Наивное сканирование сети

Выполним сканирование локального сегмента сети с параметрами по умолчанию (иначе, такой вид сканирования называют "наивным").

```
smart@thinkpad$ nmap 192.168.124.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 16:02 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0067s latency).
Not shown: 991 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
```

222/tcp open rsh-spx  
445/tcp open microsoft-ds  
1900/tcp open upnp  
8090/tcp open opsmessaging  
8200/tcp open trivnet1

Nmap scan report for 192.168.124.62

Host is up (0.00011s latency).

All 1000 scanned ports on 192.168.124.62 are in ignored states.

Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.85

Host is up (0.033s latency).

All 1000 scanned ports on 192.168.124.85 are in ignored states.

Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.96

Host is up (0.0044s latency).

Not shown: 996 closed tcp ports (conn-refused)

PORT	STATE	SERVICE
------	-------	---------

1081/tcp	open	pvuniwien
----------	------	-----------

3000/tcp	open	ppp
----------	------	-----

3001/tcp	open	nessus
----------	------	--------

9998/tcp	open	distinct32
----------	------	------------

Nmap scan report for 192.168.124.114

Host is up (0.018s latency).

All 1000 scanned ports on 192.168.124.114 are in ignored states.

Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.120

Host is up (0.0050s latency).

Not shown: 997 closed tcp ports (conn-refused)

PORT	STATE	SERVICE
------	-------	---------

49152/tcp	open	unknown
-----------	------	---------

49155/tcp	open	unknown
-----------	------	---------

62078/tcp	open	iphone-sync
-----------	------	-------------

```
Nmap scan report for 192.168.124.131
Host is up (0.0027s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
49152/tcp  open  unknown
62078/tcp  open  iphone-sync

Nmap done: 256 IP addresses (7 hosts up) scanned in 17.14 seconds
```

Итого, из 10 устройств, подключенных к роутеру, nmap смог опередить 7.

Для каждого обнаруженного хоста, Nmap пытается указать доменное имя компьютера. Далее Nmap отображает информацию о закрытых или заблокированных портах (Not shown NNNN closed ports), а затем выводит (в три колонки) порты, имеющие другой статус. Первая колонка обозначает текущий номер порта, вторая может принимать различные значения, которые будут свидетельствовать об определенном Nmap статусе порта:

- open (открытый порт) – порт открыт, и служба принимает TCP- или UDP-соединения по этому порту (данный порт наиболее уязвим для взлома);
- filtered – порт закрыт брандмауэром, иной блокирующей программой или службой (правила роутера, аппаратный брандмауэр и т.п.);
- closed – порт закрыт, так как нет службы или иной программы, прослушивающей этот порт на компьютере;
- unfiltered, open|filtered или closed|filtered – Nmap не смог точно определить, открыт порт или закрыт и необходимо использовать сканирование другим методом.

Последняя колонка делает предположение о работающем на этом порту сервисе. Допустим, если открыт порт с номером 80, Nmap полагает, что этот порт обычно применяется web-серверами (http). Но это не обязательно так – практически любой сервис может быть запущен на любом порту, и для более точного определения нужно проводить более глубокое сканирование.

## 2.2. Обнаружение компьютера методом ping

Самым простым является метод обнаружения работающих компьютеров с помощью ping

```
smart@thinkpad$ nmap -sP 192.168.124.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 16:20 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0018s latency).
Nmap scan report for 192.168.124.62
Host is up (0.000064s latency).
Nmap scan report for 192.168.124.85
Host is up (0.044s latency).
Nmap scan report for 192.168.124.104
Host is up (0.13s latency).
Nmap scan report for 192.168.124.114
Host is up (0.0034s latency).
Nmap scan report for 192.168.124.130
Host is up (0.090s latency).
Nmap scan report for 192.168.124.131
Host is up (0.084s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 5.73 seconds
```

Сетевой сканер Nmap посылает обычные ICMP запросы каждому IP-адресу из списка и ждет ответа. Если ответ получен, значит, сканируемый компьютер работает, что и отображается в качестве результата сканирования. Важно отметить, что удалённый компьютер не обязан отвечать на ICMP запрос, подобные правила легко задаются в фаерволе.

По итогам сканирования мы видим те же 7 хостов. Сканирование было выполнено гораздо быстрее, но не даёт информацию о портах.

## 2.3. Обнаружение с помощью SYN/ACK- и UDP-пакетов

Для использования флага -PU (UDP ping) требуется root-доступ, т.к. требуется доступ к сырому ответу с канального уровня.

```
smart@thinkpad$ sudo nmap -PS80 -PU -PA -PY 192.168.124.0/24
[sudo] password for smart:
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 16:27 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0020s latency).
```

Not shown: 991 closed tcp ports (reset)

PORT	STATE	SERVICE
22/tcp	open	ssh
53/tcp	open	domain
80/tcp	open	http
139/tcp	open	netbios-ssn
222/tcp	open	rsh-spx
445/tcp	open	microsoft-ds
1900/tcp	open	upnp
8090/tcp	open	opsmessaging
8200/tcp	open	trivnet1

MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.85

Host is up (0.028s latency).

All 1000 scanned ports on 192.168.124.85 are in ignored states.

Not shown: 1000 closed tcp ports (reset)

MAC Address: 34:CE:00:86:A0:C4 (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.104

Host is up (0.0047s latency).

All 1000 scanned ports on 192.168.124.104 are in ignored states.

Not shown: 1000 closed tcp ports (reset)

MAC Address: 34:CE:00:BD:11:0F (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.114

Host is up (0.0061s latency).

All 1000 scanned ports on 192.168.124.114 are in ignored states.

Not shown: 1000 closed tcp ports (reset)

MAC Address: 34:CE:00:86:67:69 (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.131

Host is up (0.0027s latency).

Not shown: 998 closed tcp ports (reset)

PORT	STATE	SERVICE
49152/tcp	open	unknown
62078/tcp	open	iphone-sync



```
MAC Address: 2E:0E:A4:CF:16:E1 (Unknown)
```

```
Nmap scan report for 192.168.124.132
```

```
Host is up (0.0064s latency).
```

```
Not shown: 997 closed tcp ports (reset)
```

```
PORT      STATE SERVICE
```

```
49152/tcp open  unknown
```

```
49153/tcp open  unknown
```

```
62078/tcp open  iphone-sync
```

```
MAC Address: D6:F0:D3:73:71:73 (Unknown)
```

```
Nmap scan report for 192.168.124.62
```

```
Host is up (0.0000040s latency).
```

```
All 1000 scanned ports on 192.168.124.62 are in ignored states.
```

```
Not shown: 1000 closed tcp ports (reset)
```

```
Nmap done: 256 IP addresses (7 hosts up) scanned in 14.89 seconds
```

Если какой-либо сервис прослушивает порт, а Nmap пытается установить соединение с ним (отсылает пакет с флагом **SYN**), в ответ сервис может послать пакет с флагами **SYN/ACK**, что покажет: компьютер в сети существует.

При отсутствии сервиса по этому порту сервер посылает в ответ пакет с флагом **RST**, что также указывает, что по заданному IP-адресу компьютер есть.

Если в ответ на посланный пакет **SYN** от сервера ничего не пришло — это значит, что либо компьютер выключен, либо трафик блокируется брандмауэром. Чтобы обойти блокирование брандмауэром, разработан еще один метод сканирования. Сканер Nmap обычно посылает пакеты с флагами **SYN/ACK** и пакет **UDP** по стандартному 80-му порту, который чаще всего используется для web-трафика и поэтому очень редко блокируется брандмауэром.

- -PS – TCP SYN
- -PA – TCP ACK
- -PU – UDP
- -PY – SCTP

## 2.4. Обнаружение компьютера посредством различных ICMP-пакетов

Ранее мы использовали простой ICMP-запрос, но такой трафик может быть заблокирован. У сетевого сканера Nmap есть другие возможности определения хоста при помощи ICMP.

```
smart@thinkpad$ sudo nmap -PE -PP -PM 192.168.124.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 16:49 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0047s latency).
Not shown: 991 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
222/tcp   open  rsh-spx
445/tcp   open  microsoft-ds
1900/tcp  open  upnp
8090/tcp  open  opsmessaging
8200/tcp  open  trivnet1
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.62
Host is up (0.000012s latency).
All 1000 scanned ports on 192.168.124.62 are in ignored states.
Not shown: 1000 closed tcp ports (reset)

Nmap done: 256 IP addresses (2 hosts up) scanned in 3.38 seconds
```

Рассмотрим дополнительные ключи:

- -PE – ICMP echo, уже использовали ранее
- -PP – ICMP timestamp requests, запрос времени
- -PM – ICMP netmask request discovery, запрос адреса и маски

С помощью этих методов тоже можно получить ответ от удаленного компьютера.

## 3. Сканирование портов

Многие методы задействуют различные манипуляции с флагами TCP-пакетов на низком уровне, а потому для работы требуют полномочий root (суперпользователя) в системе.

### 3.1. Сканирование портов методом SYN

Наиболее распространенный метод, который используется по умолчанию, — это сканирование TCP SYN. Этот подход позволяет сканировать несколько сотен портов в секунду, скрывая при этом сканирующий компьютер, поскольку никогда не завершает TCP-соединение.

```
smart@thinkpad$ sudo nmap -sS 192.168.124.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:03 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0013s latency).
Not shown: 991 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
222/tcp   open  rsh-spx
445/tcp   open  microsoft-ds
1900/tcp  open  upnp
8090/tcp  open  opsmessaging
8200/tcp  open  trivnet1
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap scan report for 192.168.124.53
Host is up (0.039s latency).
All 1000 scanned ports on 192.168.124.53 are in ignored states.
Not shown: 1000 filtered tcp ports (no-response)
MAC Address: 1C:91:80:CE:C8:F4 (Apple)

Nmap scan report for 192.168.124.96
Host is up (0.0017s latency).
Not shown: 996 closed tcp ports (reset)
```

```
PORT      STATE SERVICE
1081/tcp  open  pvuniwien
3000/tcp  open  ppp
3001/tcp  open  nessus
9998/tcp  open  distinct32
MAC Address: 38:8C:50:52:42:D7 (LG Electronics)
```

```
Nmap scan report for 192.168.124.104
Host is up (0.034s latency).
All 1000 scanned ports on 192.168.124.104 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 34:CE:00:BD:11:0F (Xiaomi Electronics,co.)
```

```
Nmap scan report for 192.168.124.114
Host is up (0.044s latency).
All 1000 scanned ports on 192.168.124.114 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 34:CE:00:86:67:69 (Xiaomi Electronics,co.)
```

```
Nmap scan report for 192.168.124.120
Host is up (0.0055s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
49152/tcp  open  unknown
49155/tcp  open  unknown
62078/tcp  open  iphone-sync
MAC Address: 6E:63:9D:97:70:7F (Unknown)
```

```
Nmap scan report for 192.168.124.62
Host is up (0.0000050s latency).
All 1000 scanned ports on 192.168.124.62 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
```

```
Nmap done: 256 IP addresses (7 hosts up) scanned in 40.58 seconds
```

Сканер Nmap отправляет исследуемому компьютеру пакет с флагом **SYN**, как будто он хочет открыть обычное TCP-соединение, следуя приведенным в начале статьи правилам.

Если ответ (пакет с флагами SYN/ACK) от запрашиваемого хоста получен, порт будет обозначен как открытый, а при получении пакета с флагом RST – как закрытый. В случае если сканируемый компьютер не ответил, предполагается, что этот порт фильтруется брандмауэром.

### 3.2. Сканирование с использованием системной функции connect()

Метод, основанный на установлении соединения с помощью системной функции connect(), которую применяет большинство приложений.

```
smart@thinkpad$ nmap -sT 192.168.124.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:03 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0076s latency).
Not shown: 991 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
222/tcp   open  rsh-spx
445/tcp   open  microsoft-ds
1900/tcp  open  upnp
8090/tcp  open  opsmessaging
8200/tcp  open  trivnet1

Nmap scan report for 192.168.124.62
Host is up (0.00014s latency).
All 1000 scanned ports on 192.168.124.62 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.85
Host is up (0.034s latency).
All 1000 scanned ports on 192.168.124.85 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.96
```

```
Host is up (0.0040s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE
1081/tcp  open  pvuniwien
3000/tcp  open  ppp
3001/tcp  open  nessus
9998/tcp  open  distinct32

Nmap scan report for 192.168.124.104
Host is up (0.0062s latency).
All 1000 scanned ports on 192.168.124.104 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.124.114
Host is up (0.0067s latency).
All 1000 scanned ports on 192.168.124.114 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap done: 256 IP addresses (6 hosts up) scanned in 26.51 seconds
```

Запрос отправляется операционной системе, которая устанавливает TCP-соединение. После определения статуса порта Nmap прерывает соединение, то есть с помощью функции `connect()` посылается пакет с флагом RST.

Отрицательной стороной является то, что подобный способ логируется в системных журналах, раскрывая факт сканирования.

### 3.3. Сканирование портов UDP-протокола

Наиболее распространенные сервисы, использующие UDP-протокол это DNS, SNMP и DHCP.

```
smart@thinkpad$ sudo nmap -sU -sV 192.168.124.1
[sudo] password for smart:
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:28 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0020s latency).
```







- `-sU` – Сам запуск UPD сканирования
- `-sV` – Определить версию запущенного сервиса (существенно увеличивает время сканирования)

### 3.4. Сканирование с помощью методов FIN, Xmas и Null

Прерывания последовательности соединения можно также получить информацию о закрытых и открытых портах исследуемого хоста.

```
smart@thinkpad$ sudo nmap -sF 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:26 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0013s latency).
All 1000 scanned ports on 192.168.124.1 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap done: 1 IP address (1 host up) scanned in 21.34 seconds

smart@thinkpad$ sudo nmap -sN 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:26 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0017s latency).
All 1000 scanned ports on 192.168.124.1 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap done: 1 IP address (1 host up) scanned in 21.29 seconds

smart@thinkpad$ sudo nmap -sX 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 17:27 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0017s latency).
All 1000 scanned ports on 192.168.124.1 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)
```

```
Nmap done: 1 IP address (1 host up) scanned in 21.27 seconds
```

Рассмотрим эти методы сканирования

- **-sF** FIN-сканирование – удаленному хосту посылаются пакеты с флагом **FIN**, которые обычно применяются при закрытии соединения. В этом случае закрытый порт компьютера, в соответствии со спецификацией протокола TCP, должен послать ответный пакет с флагом **RST**. Если же порт открыт или блокируется брандмауэром, ответа от него не будет.
- **-sN** null-сканирование – та же самая механика, но вместо пакета с **FIN**-флагом отсылается пакет с пустым заголовком (0 бит, все флаги отключены).
- **-sX** Xmas-сканирование – та же механика, но хосту отсылается пакет, раскрашенный несколькими флагами (**FIN**, **PSH** и **URG**) на манер рождественской елки.

### 3.5. Сканирование с помощью методов ACK и Window

Для определения, какие порты на компьютере находятся в статусе **filtered**, а какие в **unfiltered**, существует отдельно вынесенный тип сканирования.

```
smart@thinkpad$ sudo nmap -sA 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 18:28 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0055s latency).
All 1000 scanned ports on 192.168.124.1 are in ignored states.
Not shown: 1000 unfiltered tcp ports (reset)
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds
```

Многие фаерволы выполняют проверку только **SYN**-пакетов на определенном порту, осуществляя тем самым фильтрацию, с помощью отсылки пакетов с флагом **ACK** можно определить, существует ли на удалённом компьютере брандмауэр или нет. Пакет с флагом **ACK** в этом случае отсылается не как часть соединения, а отдельно. В случае если принимающая сторона отправляет обратный пакет с флагом **RST** (соответственно порт не

блокируется брандмауэром), порт помечается как **unfiltered**, если же хост не отвечает на пакет, то на нем установлен брандмауэр и порт находится в статусе **filtered**.

```
smart@thinkpad$ sudo nmap -sW 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 18:29 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0076s latency).
All 1000 scanned ports on 192.168.124.1 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap done: 1 IP address (1 host up) scanned in 0.39 seconds
```

Существует аналогичный метод, который работает по такому же принципу, но иначе интерпретирует полученные от хоста результаты. Сканирование методом **TCP Window** предполагает, что на некоторых хостах службы используют положительное значение поля **window** в ответном пакете (не ноль). Поэтому с помощью данного метода Nmap анализирует заголовки приходящих пакетов с флагом **RST**, и если приходящий пакет содержит положительное значение поля, то Nmap помечает этот порт открытым. Получение пакета с нулевым значением поля означает, что порт закрыт.

В нашем случае фаервол на удалённом хосте не обнаружен.

### 3.6. Сканирование методом Maimon

Способ сканирования был предложен специалистом, по имени Uriel Maimon.

Механика метода сходна с **FIN**, **Xmas** и **Null**, но отправляются пакеты с флагами **FIN/ACK**. Если порт закрыт, хост должен отвечать пакетом **RST**.

```
smart@thinkpad$ sudo nmap -sM 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 18:48 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0016s latency).
Not shown: 998 open|filtered tcp ports (no-response)
PORT      STATE SERVICE
139/tcp    closed netbios-ssn
```

```
445/tcp closed microsoft-ds
```

```
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)
```

```
Nmap done: 1 IP address (1 host up) scanned in 4.65 seconds
```

### 3.7. Скрытое сканирование с использованием алгоритма `idlescan`

Метод `idlescan` по своему алгоритму работы практически идентичен SYN-сканированию, но пытается скрыть IP-адрес хоста, с которого производится исследование.

Попытка просканировать хост 192.168.124.1 выдавая себя за хост 192.168.124.96.

```
smart@thinkpad$ sudo nmap -sI 192.168.124.96 192.168.124.1
WARNING: Many people use -Pn w/Idlescan to prevent pings from their true
↳ IP. On the other hand, timing info Nmap gains from pings can allow
↳ for faster, more reliable scans.
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 18:56 MSK
Idle scan using zombie 192.168.124.96 (192.168.124.96:443); Class:
↳ Incremental
Idle scan is unable to obtain meaningful results from proxy
↳ 192.168.124.96 (192.168.124.96). I'm sorry it didn't work out.
QUITTING!
```

Результаты такого сканирования часто бывают не точны, при этом требуют довольно много времени. В нашем случае никакие полезные результаты получить не удалось.

### 3.8. Сканирование на наличие открытых протоколов

Иногда достаточно просто определить сам факт наличия открытого порта на удалённом хосте.

Для определения доступности протокола на хосте Nmap посылает несколько пакетов с пустыми заголовками, содержащими в поле `protocol` только номер протокола. В случае, если протокол недоступен, компьютер вернет ICMP-сообщение «protocol unreachable». Если в ответ хост не посылает пакетов – это может означать, что либо протокол доступен, либо брандмауэр блокирует ICMP-трафик.

```
smart@thinkpad$ sudo nmap -s0 192.168.124.1
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 18:58 MSK
Nmap scan report for 192.168.124.1
Host is up (0.0029s latency).
Not shown: 245 closed n/a protocols (proto-unreach)

```

PROTOCOL	STATE	SERVICE
1	open	icmp
2	open filtered	igmp
4	open filtered	ipv4
6	open	tcp
17	open	udp
41	open filtered	ipv6
47	open filtered	gre
50	open filtered	esp
51	open filtered	ah
108	open filtered	ipcomp
136	open filtered	udplite

```
MAC Address: 34:CE:00:37:D9:03 (Xiaomi Electronics,co.)

Nmap done: 1 IP address (1 host up) scanned in 322.60 seconds
```

### 3.9. Скрытное сканирование посредством метода ftp bounce

Метод основан на возможности FTP-сервера отправлять файлы 3й стороне. Зная об этой уязвимости, многие владельцы FTP-серверов закрыли эту возможность.

В нашей сети нет FTP-сервера, но мы просто упоминаем эту возможность как опцию для сканирования.

```
smart@thinkpad$ sudo nmap -b <username:password@server:port>
↪ 192.168.124.1
```

## 4. Изучение служб на удалённых хостах

Nmap с большой степенью вероятности позволяет определять версию операционной системы, которая запущена на удаленном компьютере. При этом Nmap может также идентифицировать версии запущенных на удаленном ПК сервисов, при условии что порты того или иного сервиса открыты.

### 4.1. Определение версии ОС

Сканер Nmap собирает "отпечатки пальцев" (fingerprints) различных хостов и запущенных там сервисов. Сравнивая полученные результаты с эталонными значениями, указанными в файле Nmap-os-fingerprints, программа выдает сводный результат по компьютеру.

```
smart@thinkpad$ sudo nmap -O -A 192.168.124.96
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 19:24 MSK
Nmap scan report for 192.168.124.96
Host is up (0.0020s latency).
Not shown: 996 closed tcp ports (reset)
Bug in uptime-agent-info: no string output.
PORT      STATE SERVICE VERSION
1081/tcp  open  upnp      Platinum unpd 1.0.4.9 (arch: i686; UPnP 1.0;
↪ DLNADOC 1.50)
3000/tcp  open  http      LG smart TV http service
|_http-title: Site doesn't have a title.
3001/tcp  open  ssl/http  LG smart TV http service
| tls-nextprotoneg:
|   http/1.1
|_  http/1.0
|_http-title: Site doesn't have a title.
|_ssl-date: TLS randomness does not represent time
| ssl-cert: Subject:
↪ commonName=LG_TV_1ba52211ee27c01/organizationName=LG Electronics
↪ U.S.A, Inc./stateOrProvinceName=New Jersey/countryName=US
| Not valid before: 2018-11-11T07:30:39
|_Not valid after: 2038-11-06T07:30:39
9998/tcp  open  http      Google Chromecast httpd
|_http-title: Cast shell remote debugging
```

```
MAC Address: 38:8C:50:52:42:D7 (LG Electronics)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: OS: Linux; Device: media device; CPE:
↪ cpe:/o:linux:linux_kernel
```

#### TRACEROUTE

```
HOP RTT      ADDRESS
1    2.02 ms 192.168.124.96
```

```
OS and Service detection performed. Please report any incorrect results
↪ at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 108.58 seconds
```

```
smart@thinkpad$ sudo nmap -A -O 192.168.124.132
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-13 19:25 MSK
Nmap scan report for 192.168.124.132
Host is up (0.016s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
49152/tcp  open  unknown
49153/tcp  open  unknown
62078/tcp  open  iphone-sync?
MAC Address: D6:F0:D3:73:71:73 (Unknown)
No exact OS matches for host (If you know what OS is running on it, see
↪ https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=2/13%OT=49152%CT=1%CU=40622%PV=Y%DS=1%DC=D%G=Y%M=D6
↪ FOD
OS:3%TM=63EA6542%P=x86_64-pc-linux-gnu)SEQ(SP=105%GCD=1%ISR=10F%TI=Z%CI=
↪ RD%
OS:II=RI%TS=21)SEQ(SP=104%GCD=1%ISR=10D%TI=Z%CI=RD%TS=21)OPS(O1=M5B4NW6N
↪ NT1
```

```

OS: 1SLL%02=M5B4NW6NNT11SLL%03=M5B4NW6NNT11%04=M5B4NW6NNT11SLL%05=M5B4NW6
↪ NNT
OS: 11SLL%06=M5B4NNT11SLL) WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=
↪ FFF
OS: F) ECN(R=Y%DF=Y%T=40%W=FFFF%O=M5B4NW6SLL%CC=N%Q=) T1(R=Y%DF=Y%T=40%S=0%
↪ A=S
OS: +%F=AS%RD=0%Q=) T2(R=N) T3(R=N) T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0
↪ %Q=
OS: ) T5(R=Y%DF=N%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=) T6(R=Y%DF=Y%T=40%W=0%S
↪ =A%
OS: A=Z%F=R%O=%RD=0%Q=) T7(R=Y%DF=N%T=40%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=) U1(R=
↪ Y%D
OS: F=N%T=40%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=0%RUD=G) IE(R=Y%DFI=S%T
↪ =40
OS:%CD=S)

```

Network Distance: 1 hop

#### TRACEROUTE

HOP	RTT	ADDRESS
1	15.64 ms	192.168.124.132

OS and Service detection performed. Please report any incorrect results  
↪ at <https://nmap.org/submit/> .

Nmap done: 1 IP address (1 host up) scanned in 197.01 seconds

## 5. Параметры сканирования по расписанию

Nmap позволяет задать расписание сканирования, чтобы попытаться скрыть свое присутствие от брандмауэров и систем безопасности.

Существует шесть расписаний сканирования:

- **Paranoid** – с паузой в 5 минут производится тестирование хоста без ограничений на общее тестирование 5 минут таймат на ответ.
- **Sneaky** – 15 секунд между хостами, без ограничения на один хост и 15 секунд таймаут



на ответ.

- **Polite** – 0,4 секунды между хостами, без ограничений на каждый хост и 6 секунд таймат на ответ.
- **Normal** (используется по умолчанию) – без пауз между хостами и 6 секунд таймаут на запрос.
- **Insane** – без ограничений между хостами, 75 секунд на весь хост и 0,3 секунды на каждый запрос. При этом запросы могут параллеливаться.

Результаты сканирования могут быть записаны в журнал. Типы записи различаются по методу сохранения информации.

- **-oN** осуществляет запись после появления информации на экране
- **-oA** запись сразу всеми возможными форматами в файлы с названием file и различными расширениями (\*.xml, \*.gNmap, \*.Nmap)

## 6. Сравните возможности Nmap с другими средствами аудита сети

Сравнивать Nmap с Netcat или ping не корректно, т.к. это инструменты совершенно разных категорий. В известном смысле, Netcat включает в себя и ping и netcat.

Альтернативами для Nmap могут быть такие пакеты, как

- **Rapid7 Nexpose** – сканер уязвимостей, который выполняет активное сканирование ИТ-инфраструктуры на наличие ошибочных конфигураций, дыр, вредоносных кодов, и предоставляет рекомендации по их устранению. Под анализ попадают все компоненты инфраструктуры, включая сети, операционные системы, базы данных и web-приложения. По результатам проверки Rapid7 Nexpose в режиме приоритетов классифицирует обнаруженные угрозы и генерирует отчеты по их устранению. [www.rapid7.com](http://www.rapid7.com)
- **Tenable Nessus Scanner** – сканер, предназначенный для оценки текущего состояния защищенности традиционной ИТ-инфраструктуры, мобильных и облачных сред, контейнеров и т.д. По результатам сканирования выдаёт отчёт о найденных уязвимостях. Рекомендуется использовать, как составную часть Nessus Security Center. [www.tenable.com/products/nessus/nessus-professional](http://www.tenable.com/products/nessus/nessus-professional)

- OpenVAS – сканер уязвимостей с открытым исходным кодом. OpenVAS предназначен для активного мониторинга узлов вычислительной сети на предмет наличия проблем, связанных с безопасностью, оценки серьезности этих проблем и для контроля их устранения. Активный мониторинг означает, что OpenVAS выполняет какие-то действия с узлом сети: сканирует открытые порты, посылает специальным образом сформированные пакеты для имитации атаки или даже авторизуется на узле, получает доступ к консоли управления, и выполняет на нем команды. Затем OpenVAS анализирует собранные данные и делает выводы о наличии каких-либо проблем с безопасностью. Эти проблемы, в большинстве случаев касаются установленного на узле необновленного ПО, в котором имеются известные и описанные уязвимости, или же небезопасно настроенного ПО. [www.openvas.org](http://www.openvas.org)

Дальнейшее сравнение целесообразно проводить при наличии критериев сравнения, которые можно было бы выделить из постановки задачи (хотим ли мы наземно анализировать чужую сеть в рамках пинтеста, хотим ли мы отслеживать актуальность ПО в своей сети, либо что-то ещё).

## Выводы

Виды сканирования различаются по скорости и полноте исследования. Комбинируя их, можно получить максимальное представление о сети, её хостах и работающих на этих хостах сервисах.

В процессе исследования легче всего поддавались изучению хосты на базе linux, хуже всего – устройства от Apple. Возможно это связано с тем, что в базе данных отпечатков не было данных для новых устройств.

# Лабораторная работа 7. Утилиты Netcat и Cryptcat

**Цель работы:** Исследовать взаимодействие процессов на разных хостах при помощи утилит Netcat и её защищенного аналога Cryptcat.

## 1. Установка

Утилита Netcat уже была установлена в рамках предыдущих работ для эмуляции Telnet подключения

```
smart@thinkpad$ pacman -Qi netcat
Name           : gnu-netcat
Version        : 0.7.1-9
Description     : GNU rewrite of netcat, the network piping application
Architecture   : x86_64
URL            : http://netcat.sourceforge.net/
Licenses       : GPL
Groups         : None
Provides       : netcat
Depends On     : glibc texinfo
Optional Deps  : None
Required By    : None
Optional For   : None
Conflicts With : None
Replaces       : netcat
Installed Size : 66.24 KiB
Packager       : Felix Yan <felixonmars@archlinux.org>
Build Date    : Tue 27 Dec 2022 02:47:12 PM MSK
```

Install Date	: Wed 08 Feb 2023 11:43:18 PM MSK
Install Reason	: Explicitly installed
Install Script	: No
Validated By	: Signature

## 2. Взаимодействие процессов

Организуем простейшее сетевое взаимодействие двух процессов.

В этом и дальнейших примерах в качестве адреса будет использован адрес локального интерфейса, но на практике можно использовать любой маршрутизируемый адрес.

Процесс-сервер будет работать в режиме прослушивания. Для его запуска, будем использовать команду `nc -nvlp 1234`, где:

- `-n` – не выполнять резолвинг имён через DNS
- `-v` – подробный (verbose) режим отображения логов
- `-l` – режим прослушивания (в некоторых реализациях netcat, `-l` завершит сервер после разрыва соединения, `-L` перезапустит сервер)
- `-p 1234` – номер порта, на котором ожидается соединение

Опционально, можно указать `-t` для TCP соединения (используется по умолчанию) или `-u` для UDP.

Запуск сервера и общение с клиентом

```
smart@thinkpad$ nc -nvlp 1234
Connection from 127.0.0.1:52880
Hi from client
Hi from server
```

Для запуска клиента будем использовать команду `nc 127.0.0.1 1234`, где:

- `127.0.0.1` – IP-адрес сервера
- `1234` – порт, на котором слушает сервер

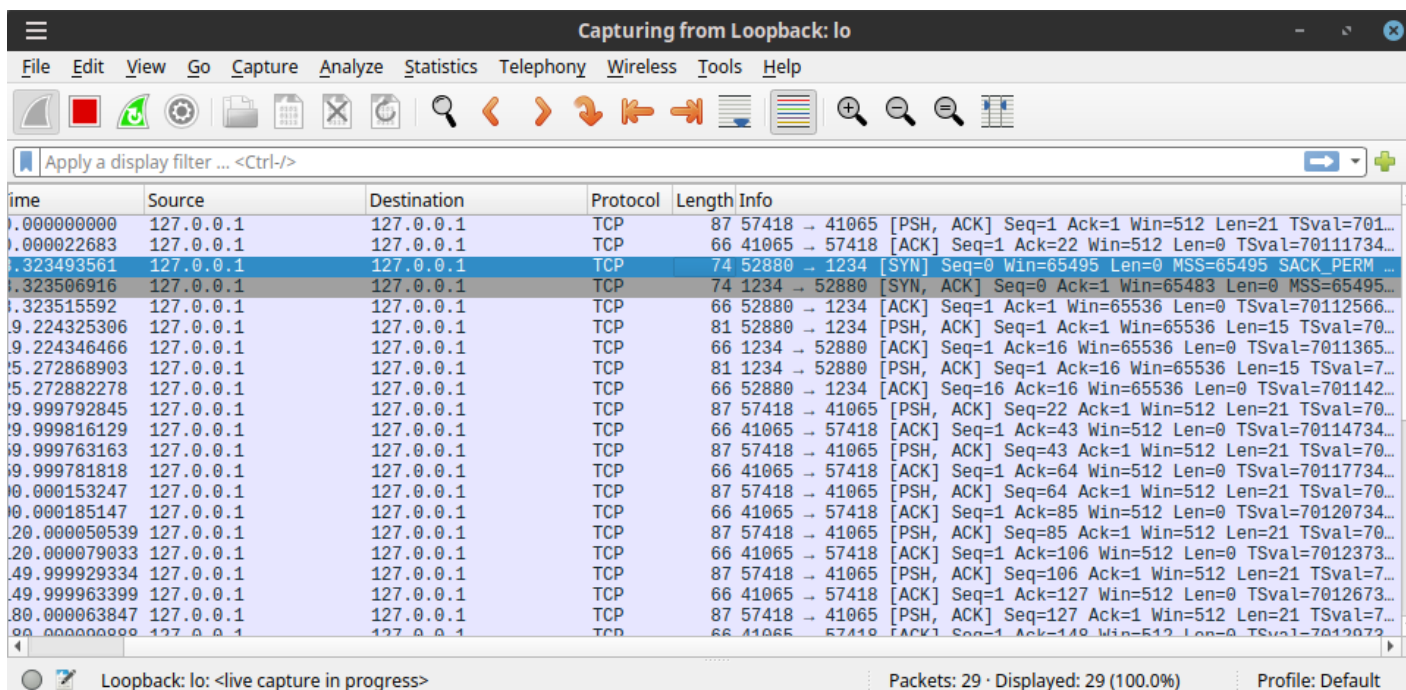
## Запуск клиента и общение с сервером

```
smart@thinkpad$ nc 127.0.0.1 1234
Hi from client
Hi from server
```

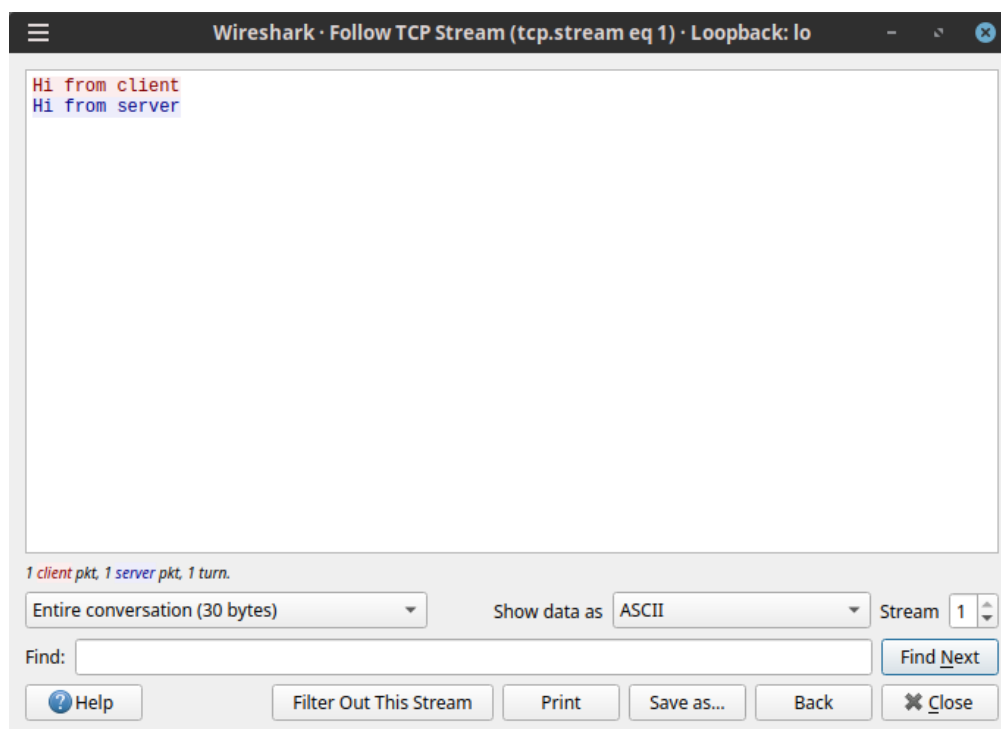
## Проверка состояние сокета

```
smart@thinkpad$ ss -tp | grep nc
ESTAB 0    0        127.0.0.1:52880          127.0.0.1:search-agent
↪ users:(("nc",pid=44033,fd=3))
ESTAB 0    0        127.0.0.1:search-agent  127.0.0.1:52880
↪ users:(("nc",pid=43782,fd=4))
```

## Пакеты, пойманные Wireshark



Wireshark позволяет полностью отследить всю переписку



## 3. Передача файлов

Все следующие примеры (включая сжатие) подразумевают не зашифрованную передачу, это значит Wireshark может перехватить и восстановить передаваемые файлы.

### 3.1 Передача одного файла

На принимающей стороне, запускаем прослушивающий сервер `nc -nvlp 1234 > file.txt`.

Все параметры запуска уже были рассмотрены в предыдущем примере, вывод команды сделан в файл `file.txt`.

Получение файла

```
smart@thinkpad$ nc -nvlp 1234 > file.txt
Connection from 127.0.0.1:56638
```

На передающей стороне, делаем такой вызов `cat file.txt | nc 127.0.0.1 1234`. В некоторых реализациях есть дополнительная опция `-q 0` для того, что бы netcat автоматически завершил работу сразу после отправки.

Передача файла

```
smart@thinkpad$ nc 127.0.0.1 1234 < file.txt
```

## 3.2 Передача одного файла с отображением прогресса

Главный минус предыдущего подхода в том, что непонятно когда завершена передача.

Возможным решением может быть добавление счётчика на отправляющий и принимающей стороне.

Получение файла

```
smart@thinkpad$ nc -nvlp 1234 | pv -b > file.txt
Connection from 127.0.0.1:36388
304 B
```

Передача файла

```
smart@thinkpad$ cat file.txt | pv -b | nc 127.0.0.1 1234
304 B
```

При помощи утилиты `pv` (pipeviewer) мы можем видеть прогресс передачи через подсчёт количества байт (ключ `-b` просто отображает итоговый объём, отключая анимацию передачи, хотя она бывает полезна при передаче больших объёмов).

## 3.3 Передача нескольких файлов

Для передачи нескольких файлов в одной директории можно написать простой `bash`-скрипт, который переберёт все файлы, и для каждого сделает вызов `nc`.

Другой вариант передачи – запаковать её в `tar`бол на одном конце, и распаковать на другом. Кроме прочего, это позволит передать и права, выставленные на файлы.

Получение директории

```
smart@thinkpad$ nc -nvlp 1234 | pv -b | tar xf -  
Connection from 127.0.0.1:35942  
39.0MiB
```

Тут мы используем утилиту tar:

- **x** – производить операцию извлечения из обрабатываемого потока
- **f** – обрабатывать архивный файл
- **-** – вместо имени архива поток, полученный из пайпа

Передача директории

```
smart@thinkpad$ tar cf - . | pv -b | nc 127.0.0.1 1234  
39.0MiB
```

Тут мы используем утилиту tar:

- **c** – производить операцию сжатия
- **f** – обрабатывать архивный файл
- **-** – вместо файла, передаём данные в пайп
- **.** – обрабатываем все файлы в текущей директории

### 3.4 Передача нескольких файлов со сжатием

Для экономии трафика, можно сразу сжимать поток любым архиватором. Тема выбора подходящего архиватора для различных типов данных достойна отдельного исследования, мы будем использовать gzip, как оптимальное решение между скоростью работы, качеством сжатия и количеством потребляемых ресурсов. Для этого достаточно передать параметр **z** утилите tar.

Получение директории со сжатием



```
smart@thinkpad$ nc -nvlp 1234 | pv -b | tar xzf -  
Connection from 127.0.0.1:43324  
36.8MiB
```

Передача директории

```
smart@thinkpad$ tar czf - . | pv -b | nc 127.0.0.1 1234  
36.8MiB
```

Удалось уменьшить объём передаваемого трафика с 39.0MiB до 36.8MiB (экономия больше 5%).

### 3.5 Передача образа жёского диска

Иногда необходимо сделать резервную копию образа жёсткого диска с одной машины на другую.

В таком случае, для отправки образа можно использовать такую команду `bzip2 -c /dev/sdb1 | nc -nvlp 1234`. Эта команда будет пересылать все данные с диска `/dev/sdb1`. Учитывая, что мы читаем диск как набор секторов (в двоичном формате), для сжатия будем использовать `bzip2`.

Со стороны получателя, будет распаковывать поток и складывать в отдельный файл `nc 192.168.1.102 1234 | pv -b | bzip2 -d > hdImage.img`

## 4. Защищенное взаимодействие

### 4.1 Установка cryptcat

Утилита `cryptcat` довольно специфичная. Её нет в стандартном репозитории для моего дистрибутива `linux` (`Arch linux`) и нет а пользовательских пакетах (`AUR`). Дистрибутив `Kali linux` использует довольно старую версию пакета (20031202), но на официальном сайте (<https://cryptcat.sourceforge.net/>) есть ссылка на `sourceforge`, где размещена версия от октября 2005-го года.

Берём исходники утилиты по ссылке <https://sourceforge.net/projects/cryptcat/files/cryptcat-unix-1.2/cryptcat-unix-1.2.1/cryptcat-unix-1.2.1.tar/download>

Изучение исходников показывает, что по сути это старая версия утилиты Netcat для которой имплантирован модуль twofish2. Примечания к релизу содержат некоторое количество известных багов, которые планируется решить когда-то в будущем.

Сборка из исходников сыпет большим количеством предупреждений, т.к. код был написан на более старую версию API glibc.

## 4.2 Взаимодействие процессов

Повторим эксперимент с созданием чата.

Запуск сервера и общение с клиентом

```
smart@thinkpad$ ./cryptcat -nvlp 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 39944
hello from client
hello from server
```

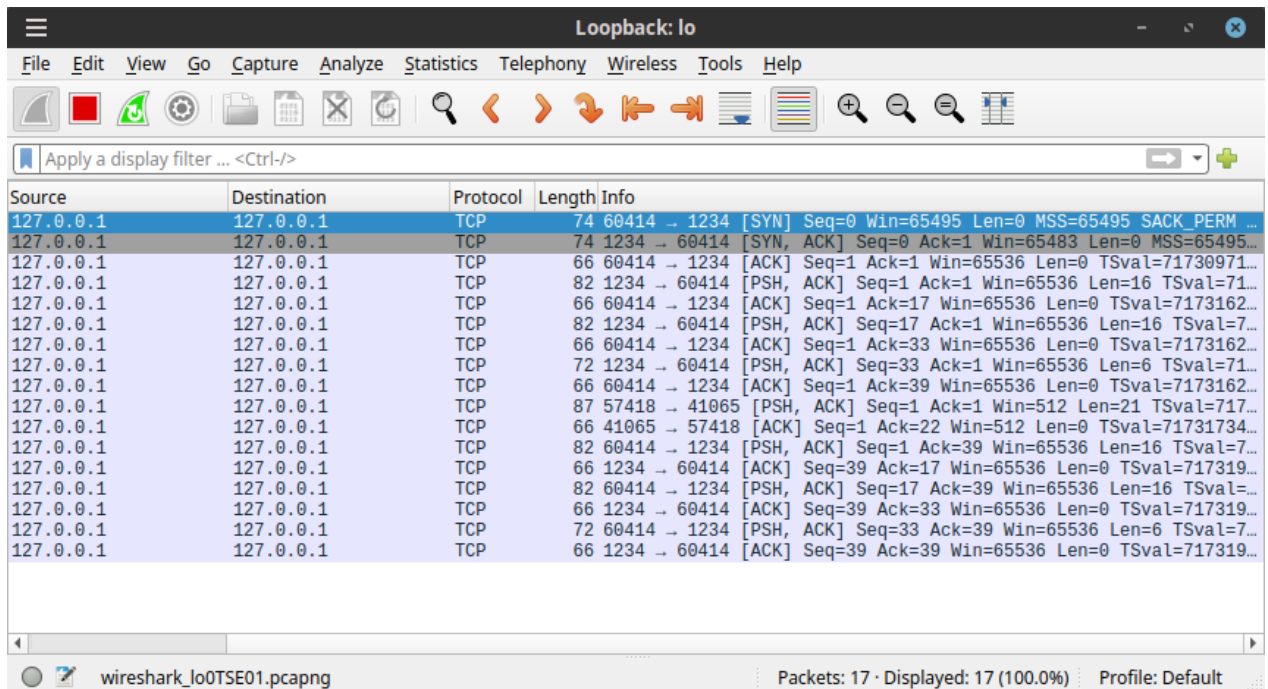
Запуск клиента и общение с сервером

```
smart@thinkpad$ ./cryptcat 127.0.0.1 1234
hello from client
hello from server
```

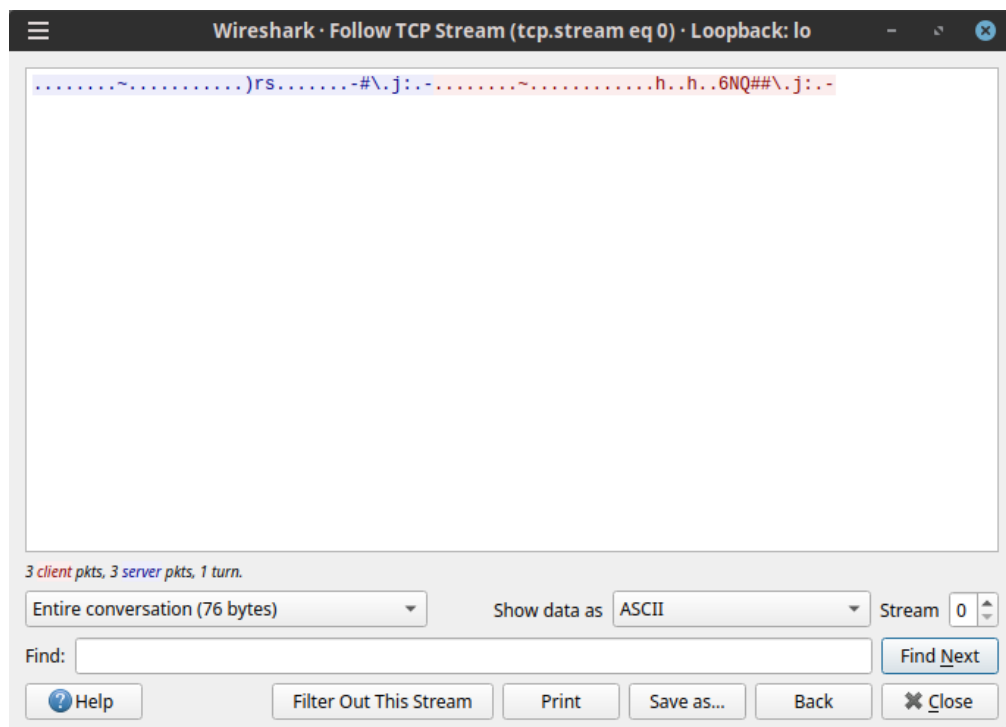
Проверка состояние сокета

```
smart@thinkpad$ ss -tp | grep nc
ESTAB 0    0        127.0.0.1:39944          127.0.0.1:search-agent
↪  users:(("cryptcat",pid=71954,fd=3))
ESTAB 0    0        127.0.0.1:search-agent  127.0.0.1:39944
↪  users:(("cryptcat",pid=71953,fd=4))
```

## Пакеты, пойманные Wireshark



Wireshark показывает, что данные переданы в зашифрованном виде



## 4.3 Передача файлов

Передача файлов работает аналогично Netcat, данные зашифрованы.

Получение файла

```
smart@thinkpad$ ../cryptcat -nvlp 1234 | pv -b > file.txt
listening on [any] 1234 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 54856
304 B
```

Передача файла

```
smart@thinkpad$ cat file.txt | pv -b | ../cryptcat 127.0.0.1 1234
304 B
```

## 5. Direct Network Traffic

Выполним установку двунаправленного пайпа для редиректа локального порта 1234 на 80-й порт google.

Для начала определим, что должно получиться в итоге. При запросе на 80-й порт, google отвечает 301 и отправляет устанавливать https соединение. Но даже такого ответа нам достаточно для теста.

```
smart@thinkpad$ curl google.com
<HTML><HEAD><meta http-equiv="content-type"
↪  content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

Подготовим двунаправленный пайп. `nc -l -p 1233 | nc www.google.com 80 | nc -l -p 1234`

После этого, через curl вначале сделаем запрос на порт 1233, потом прочитаем ответ с порта 1234.

Результат работы пайпа

```
smart@thinkpad$ nc -lp 1233 | nc google.com 80 | nc -lp 1234
GET / HTTP/1.1
Host: 127.0.0.1:1234
User-Agent: curl/7.87.0
Accept: */*
```

Результат работы curl

```
smart@thinkpad$ curl 127.0.0.1:1233
^[[A^C
smart@thinkpad$ curl 127.0.0.1:1234
<HTML><HEAD><meta http-equiv="content-type"
↪  content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com:1233/">here</A>.
</BODY></HTML>
```

Естественно, весь трафик ходит незащищенным.

## Выводы

В этой работе мы познакомились с возможностями утилиты netcat и её безопасной (но, не поддерживаемой) версии cryptcat.

С практической точки зрения, для передачи файлов удобнее использовать rsync (он безопасный и имеет множество полезных опций), безопасно туннелировать трафик можно при помощи ssh, а локальный редирект портов можно осуществлять средствами встроенного фаервола.

# Лабораторная работа 8. Сетевое экранирование. Применение правил iptables

Цель работы:

## 1. Определение IP-адреса

Выполним подключение по ssh к удалённому хосту и запросим его IP адрес с помощью утилиты ip.

```
user@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    ↪ group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    ↪ state UP group default qlen 1000
    link/ether 08:00:27:2a:f7:85 brd ff:ff:ff:ff:ff:ff
    inet 192.168.124.45/24 metric 100 brd 192.168.124.255 scope global
        ↪ dynamic enp0s3
    valid_lft 24307sec preferred_lft 24307sec
    inet6 fe80::a00:27ff:fe2a:f785/64 scope link
    valid_lft forever preferred_lft forever
```

Интересующий нас адрес – 192.168.124.45.

## 2. Просмотр текущие правил

Список текущих правил можно получить командой

`iptables -list -numeric -verbose -line-numbers` (требуется прав суперпользователя),  
где:

- `-list` – показать все правила в выбранной цепочке (если цепочка не выбрана, то во всех цепочках);
- `-numeric` – выводить IP-адреса и номера портов в числовом виде предотвращая попытки преобразовать их в символические имена;
- `-verbose` – увеличить подробность сообщений (имена интерфейсов, параметры правил, маски TOS, счётчики);
- `-line-numbers` – вывод номера строк соответствующих позиции правила в цепочке.

```
user@ubuntu:~$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source    destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source    destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source    destination
```

На данном этапе все цепочки пусты.

## 3. Блокировка входящего трафика

Заблокируем весь входящий трафик, модифицировав таблицу INPUT

```
user@ubuntu:~$ sudo iptables --append INPUT --jump DROP
```

Это естественным образом привело к "зависанию" терминала и обрыву сессии по таймауту.

Новое соединение так же установить нельзя, оно отваливается по таймауту.

```
smart@thinkpad$ ssh user@192.168.124.45 -vvvv
OpenSSH_9.2p1, OpenSSL 3.0.8 7 Feb 2023
debug1: Reading configuration data /home/smart/.ssh/config
debug1: /home/smart/.ssh/config line 9: Applying options for *
debug1: Reading configuration data /etc/ssh/ssh_config
debug2: resolve_canonicalize: hostname 192.168.124.45 is address
debug3: expanded UserKnownHostsFile '~/.ssh/known_hosts' ->
    ↪ '/home/smart/.ssh/known_hosts'
debug3: expanded UserKnownHostsFile '~/.ssh/known_hosts2' ->
    ↪ '/home/smart/.ssh/known_hosts2'
debug1: auto-mux: Trying existing master
debug1: Control socket "/tmp/ssh-master-socket-user@192.168.124.45:22"
    ↪ does not exist
debug3: ssh_connect_direct: entering
debug1: Connecting to 192.168.124.45 [192.168.124.45] port 22.
debug3: set_sock_tos: set socket 3 IP_TOS 0x48
debug1: connect to address 192.168.124.45 port 22: Connection timed out
ssh: connect to host 192.168.124.45 port 22: Connection timed out
```

Запросы ping до удалённого хоста не проходят

```
smart@thinkpad$ ping 192.168.124.45
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
^C
--- 192.168.124.45 ping statistics ---
283 packets transmitted, 0 received, 100% packet loss, time 285758ms
```

Сканер Nmap не может обнаружить открытые порты (но видит MAC-адрес сетевого интерфейса)

```
smart@thinkpad$ sudo nmap -s0 192.168.124.45
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-16 12:29 MSK
Nmap scan report for 192.168.124.45
```



```
Host is up (0.00020s latency).
All 256 scanned ports on 192.168.124.45 are in ignored states.
Not shown: 256 open|filtered n/a protocols (no-response)
MAC Address: 08:00:27:2A:F7:85 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 6.40 seconds
```

С удалённого хоста тоже не получается обратиться ко внешней сети

```
user@ubuntu$ ping ya.ru
^C
~

user@ubuntu$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4080ms
```

Теперь посмотрим правила iptables на удалённой машине, и отметим увеличение счётчика заблокированных пакетов.

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination
1      267 65107 DROP      all  --  *   *   0.0.0.0/0   0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination
```

## 4. Фильтрация входящего трафика

Тут мы предполагаем, что на удалённом сервере работают следующие сервисы:

- DNS: 53/udp – стандартный порт
- DNS: 53/tcp – RFC описывает случаи, когда DNS может/должен переходить на TCP
- WEB: 80/tcp – стандартный порт http
- WEB: 443/tcp – стандартный порт https
- WEB: 443/udp – стандартный порт https (http3)

Установим DROP в качестве политики по умолчанию:

```
sudo iptables -policy INPUT DROP
```

Разрешим свободное хождение трафика на локальном интерфейсе:

```
sudo iptables -append INPUT -in-interface lo -jump ACCEPT
```

Разрешим DNS подключения:

```
sudo iptables -append INPUT -protocol udp -dport 53 -jump ACCEPT
```

```
sudo iptables -append INPUT -protocol tcp -dport 53 -jump ACCEPT
```

Ответы, от вышестоящих DNS-серверов:

```
sudo iptables -append INPUT -protocol udp -sport 53 -dport 1024:65535 -jump ACCEPT
```

```
sudo iptables -append INPUT -protocol tcp -sport 53 -dport 1024:65535 -jump ACCEPT
```

Запросы для Web-сервера:

```
sudo iptables -append INPUT -protocol tcp -match multiport -dports 80,443 -jump ACCEPT
```

```
sudo iptables -append INPUT -protocol udp -dport 443 -jump ACCEPT
```

Итоговые правила выглядят следующим образом

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy DROP 0 packets, 0 bytes)
num pkts bytes target    prot opt in  out  source      destination
1      0      0 ACCEPT  all  --  lo   *    0.0.0.0/0   0.0.0.0/0
2      0      0 ACCEPT  udp  --  *    *    0.0.0.0/0   0.0.0.0/0   udp
↳ dpt:53
3      0      0 ACCEPT  tcp  --  *    *    0.0.0.0/0   0.0.0.0/0   tcp
↳ dpt:53
4      0      0 ACCEPT  udp  --  *    *    0.0.0.0/0   0.0.0.0/0   udp
↳ spt:53 dpts:1024:65535
4      17   3170 ACCEPT  udp  --  *    *    0.0.0.0/0   0.0.0.0/0   udp
↳ spt:53 dpts:1024:65535
```

```

5      0      0 ACCEPT  tcp  --  *   *   0.0.0.0/0  0.0.0.0/0  tcp
↪ spt:53 dpts:1024:65535
6      7    366 ACCEPT  tcp  --  *   *   0.0.0.0/0  0.0.0.0/0
↪ multiport dports 80,443
7      0      0 ACCEPT  udp   --  *   *   0.0.0.0/0  0.0.0.0/0  udp
↪ dpt:443

```

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)

num	pkts	bytes	target	prot	opt	in	out	source	destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)

num	pkts	bytes	target	prot	opt	in	out	source	destination

Для эмуляции работы WEB-сервера, используем Netcat.

На стороне сервера:

```

user@ubuntu$ sudo nc -l 80
1
2
3

```

На стороне клиента:

```

smart@thinkpad$ nc 192.168.124.45 80
1
2
3

```

Передача происходит без потерь.

## Выводы

В этой работе мы познакомились с основными возможностями по управлению цепочками в iptables.

По итогам работы, мы получили сервер, защищённый от внешних подключений.

# Лабораторная работа 9. Сетевое экранирование. Работа с iptables

Цель работы:

## 1. Предусловия

Удалённый сервер с чистыми цепочками iptables

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination
```

## 2. Запрет ICMP ping запросов извне

Запретим удалённому хосту принимать ICMP ping запросы:

```
sudo iptables -append INPUT -protocol icmp -icmp-type echo-request -jump DROP
```

Состояние iptables цепочек после выполнение предыдущей команды

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
num  pkts bytes target    prot opt in  out  source      destination
```

```

1          6   504 DROP          icmp -- *   *   0.0.0.0/0 0.0.0.0/0
↳ icmp type 8

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in   out   source destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in   out   source destination

```

Запрос извне

```

smart@thinkpad$ ping 192.168.124.45
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
^C
--- 192.168.124.45 ping statistics ---
100 packets transmitted, 0 received, 100% packet loss, time 100325ms

```

Локальный запрос так же отсеян, т.к. запросы обрабатываются общей цепочкой

```

user@ubuntu$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
^C
--- localhost ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9940ms

```

Исправить эту ситуацию можно, допустим, такой командой.

```
sudo iptables -insert INPUT -in-interface lo -jump ACCEPT
```

Состояние правил

```

user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in   out   source destination
1      18  1512 ACCEPT    all  --  lo   *    0.0.0.0/0 0.0.0.0/0
2      33  2772 DROP      icmp -- *    *    0.0.0.0/0 0.0.0.0/0
↳ icmp type 8

```

```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source destination
```

Важно отметить, что разрешение для localhost было добавлено перед запрещающим правилом.

После этого локальный запрос начинает работать

```
user@ubuntu$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
^C
--- localhost ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9940ms
```

### 3. Разрешение ICMP запросов

В этом сценарии, политикой по умолчанию для INPUT и OUTPUT цепочек будет DROP:

```
sudo iptables -policy INPUT DROP
```

```
sudo iptables -policy OUTPUT DROP
```

В таком случае, потребуется явно разрешить как получение запросов, так и ответ на них:

```
sudo iptables -insert INPUT -in-interface enp0s3 -protocol icmp -icmp-type 8 -source
0/0 -destination 192.168.124.45 -match state -state NEW,ESTABLISHED,RELATED -jump
ACCEPT
```

```
sudo iptables -insert OUTPUT -protocol icmp -icmp-type 0 -source 192.168.124.45
-destination 0/0 -match state -state ESTABLISHED,RELATED -jump ACCEPT
```

Состояние цепочек правил

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy DROP 74 packets, 19555 bytes)
```

```

num  pkts bytes target  prot opt in  out  source
↪ destination
1      0      0 ACCEPT  icmp --  enp0s3 *  0.0.0.0/0
↪ 192.168.124.45  icmp type 8 state NEW,RELATED,ESTABLISHED

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target  prot opt in  out  source
↪ destination

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
num  pkts bytes target  prot opt in  out  source
↪ destination
1      0      0 ACCEPT  icmp --  *  *  192.168.124.45  0.0.0.0/0
↪ icmp type 0 state RELATED,ESTABLISHED

```

В результате, хост отвечает на ping

```

smart@thinkpad$ ping 192.168.124.45
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
64 bytes from 192.168.124.45: icmp_seq=1 ttl=64 time=0.197 ms
64 bytes from 192.168.124.45: icmp_seq=2 ttl=64 time=0.149 ms
64 bytes from 192.168.124.45: icmp_seq=3 ttl=64 time=0.265 ms
64 bytes from 192.168.124.45: icmp_seq=4 ttl=64 time=0.233 ms
64 bytes from 192.168.124.45: icmp_seq=5 ttl=64 time=0.178 ms
64 bytes from 192.168.124.45: icmp_seq=6 ttl=64 time=0.216 ms
64 bytes from 192.168.124.45: icmp_seq=7 ttl=64 time=0.268 ms
^C
--- 192.168.124.45 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6091ms
rtt min/avg/max/mdev = 0.149/0.215/0.268/0.040 ms

```

## 4. Ограничение количества запросов

Попробуем ограничить число ICMP запросов – 1 запрос в секунду:

```
sudo iptables -append INPUT -protocol icmp -match icmp -icmp-type 8 -match limit
```

```
-limit 3/minute -limit-burst 5 -jump ACCEPT
```

Очередь будет разгружаться каждые 20 секунд.

Политикой по умолчанию для INPUT будет DROP:

```
sudo iptables -policy INPUT DROP
```

Состояние цепочек правил

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy DROP 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source
    ↪ destination
1      0    0 ACCEPT    icmp -- *      *      0.0.0.0/0    0.0.0.0/0
    ↪ icmp_type 8 limit: avg 3/min burst 5

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source
    ↪ destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out     source
    ↪ destination
```

Клиент отправляет запросы каждую 1/10 секунды, в итоге много потерь.

```
smart@thinkpad$ ping 192.168.124.45 -i 0.1
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
64 bytes from 192.168.124.45: icmp_seq=84 ttl=64 time=0.194 ms
64 bytes from 192.168.124.45: icmp_seq=270 ttl=64 time=0.268 ms
64 bytes from 192.168.124.45: icmp_seq=456 ttl=64 time=0.268 ms
64 bytes from 192.168.124.45: icmp_seq=642 ttl=64 time=0.257 ms
^C
--- 192.168.124.45 ping statistics ---
775 packets transmitted, 4 received, 99.4839% packet loss, time 83204ms
rtt min/avg/max/mdev = 0.194/0.246/0.268/0.030 ms
```



## 5. Блокировка входящих запросов

Проверим параметры блокировки входящих запросов.

### 5.1 Блокировка по адресу

Заблокируем все входящие запросы с определенного адреса (например, 192.168.124.62):

```
sudo iptables -append INPUT -source 192.168.124.62 -jump DROP
```

Состояние цепочек правил

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 1229 packets, 231K bytes)
num  pkts bytes target    prot opt in  out  source
↳ destination
1      4   336 DROP      all  --  *   *    192.168.124.62  0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source
↳ destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source
↳ destination
```

Ping с 192.168.124.62 на 192.168.124.45 не проходит.

```
smart@thinkpad$ ping 192.168.124.45 -i 0.1
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
64 bytes from 192.168.124.45: icmp_seq=84 ttl=64 time=0.194 ms
64 bytes from 192.168.124.45: icmp_seq=270 ttl=64 time=0.268 ms
64 bytes from 192.168.124.45: icmp_seq=456 ttl=64 time=0.268 ms
64 bytes from 192.168.124.45: icmp_seq=642 ttl=64 time=0.257 ms
^C
--- 192.168.124.45 ping statistics ---
775 packets transmitted, 4 received, 99.4839% packet loss, time 83204ms
rtt min/avg/max/mdev = 0.194/0.246/0.268/0.030 ms
```

## 5.2 Блокировка по порту

Заблокируем все входящие запросы с определенного адреса (например, 192.168.124.62):

```
sudo iptables -append INPUT -protocol tcp -dport 80 -jump DROP
```

Состояние цепочек правил

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination
1      0    0 DROP      tcp  --  *   *   0.0.0.0/0    0.0.0.0/0
    ↪ tcp dpt:80

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in  out  source      destination
```

На сервере запускаем Netcat на 80-ом порту.

```
user@ubuntu$ sudo nc -l 80
```

С клиента проходит ping

```
smart@thinkpad$ ping 192.168.124.45
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
64 bytes from 192.168.124.45: icmp_seq=1 ttl=64 time=0.214 ms
64 bytes from 192.168.124.45: icmp_seq=2 ttl=64 time=0.227 ms
64 bytes from 192.168.124.45: icmp_seq=3 ttl=64 time=0.252 ms
64 bytes from 192.168.124.45: icmp_seq=4 ttl=64 time=0.224 ms
^C
--- 192.168.124.45 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3041ms
rtt min/avg/max/mdev = 0.214/0.229/0.252/0.014 ms
```

Но подключиться на 80-й порт не получится

```
smart@thinkpad$ nc 192.168.124.45 80
```

А на стороне сервера увеличивается число пакетов, попавших под правило о блокировке.

### 5.3 Блокировка по адресу и порту

```
sudo iptables -append INPUT -protocol tcp -source 192.168.124.62 -dport 80 -jump DROP
```

Демонстрация смысла не имеет, т.к. наблюдаемое поведение аналогично предыдущему примеру.

### 5.4 Блокировка по MAC адресу

```
sudo iptables -append INPUT -match mac -mac-source 00:0F:EA:91:04:08 -jump DROP
```

Демонстрация смысла не имеет, т.к. наблюдаемое поведение аналогично предыдущему примеру.

## 6. Разрешение соединений только для протокола TCP

Политикой по умолчанию для INPUT будет DROP:

```
sudo iptables -policy INPUT DROP
```

Команда, которая разрешит SSH соединения с указанного MAC-адреса:

```
sudo iptables -append INPUT -protocol tcp -destination-port 22 -match mac -mac-source 00:0F:EA:91:04:08 -j ACCEPT
```

Состояние цепочек правил

```
user@ubuntu$ sudo iptables --list --numeric --verbose --line-numbers
Chain INPUT (policy DROP 2 packets, 208 bytes)
num  pkts bytes target    prot opt in      out     source       destination
1      0     0 ACCEPT    tcp  --  *      *       0.0.0.0/0    0.0.0.0/0
    ↪      tcp dpt:22 MAC00:0f:ea:91:04:08
```

```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out   source      destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes target    prot opt in     out   source      destination
```

Ping с клиента не проходит

```
smart@thinkpad$ ping 192.168.124.45
PING 192.168.124.45 (192.168.124.45) 56(84) bytes of data.
^C
--- 192.168.124.45 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2041ms
```

A SSH соединение работает

```
smart@thinkpad$ ssh user@192.168.124.45 -v
OpenSSH_9.2p1, OpenSSL 3.0.8 7 Feb 2023
debug1: Reading configuration data /home/smart/.ssh/config
debug1: /home/smart/.ssh/config line 9: Applying options for *
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: auto-mux: Trying existing master
debug1: Control socket "/tmp/ssh-master-socket-user@192.168.124.45:22"
↳ does not exist
debug1: Connecting to 192.168.124.45 [192.168.124.45] port 22.
debug1: Connection established.
debug1: identity file /home/smart/.ssh/id_rsa type 0
debug1: identity file /home/smart/.ssh/id_rsa-cert type -1
debug1: identity file /home/smart/.ssh/id_ecdsa type -1
debug1: identity file /home/smart/.ssh/id_ecdsa-cert type -1
debug1: identity file /home/smart/.ssh/id_ecdsa_sk type -1
debug1: identity file /home/smart/.ssh/id_ecdsa_sk-cert type -1
debug1: identity file /home/smart/.ssh/id_ed25519 type -1
debug1: identity file /home/smart/.ssh/id_ed25519-cert type -1
debug1: identity file /home/smart/.ssh/id_ed25519_sk type -1
debug1: identity file /home/smart/.ssh/id_ed25519_sk-cert type -1
debug1: identity file /home/smart/.ssh/id_xmss type -1
```

```
debug1: identity file /home/smart/.ssh/id_xmss-cert type -1
debug1: identity file /home/smart/.ssh/id_dsa type -1
debug1: identity file /home/smart/.ssh/id_dsa-cert type -1
debug1: Local version string SSH-2.0-OpenSSH_9.2
debug1: Remote protocol version 2.0, remote software version
↳ OpenSSH_9.0p1 Ubuntu-1ubuntu7.1
debug1: compat_banner: match: OpenSSH_9.0p1 Ubuntu-1ubuntu7.1 pat
↳ OpenSSH* compat 0x04000000
debug1: Authenticating to 192.168.124.45:22 as 'user'
debug1: load_hostkeys: fopen /home/smart/.ssh/known_hosts2: No such file
↳ or directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts: No such file or
↳ directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts2: No such file or
↳ directory
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: sntrup761x25519-sha512@openssh.com
debug1: kex: host key algorithm: ssh-ed25519
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC:
↳ <implicit> compression: zlib@openssh.com
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC:
↳ <implicit> compression: zlib@openssh.com
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: SSH2_MSG_KEX_ECDH_REPLY received
debug1: Server host key: ssh-ed25519
↳ SHA256:VkxEMLu05eDeu12IbmBEnWIEKf/4SuRitIGVjYeH6z0
debug1: load_hostkeys: fopen /home/smart/.ssh/known_hosts2: No such file
↳ or directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts: No such file or
↳ directory
debug1: load_hostkeys: fopen /etc/ssh/ssh_known_hosts2: No such file or
↳ directory
debug1: Host '192.168.124.45' is known and matches the ED25519 host key.
debug1: Found key in /home/smart/.ssh/known_hosts:58
debug1: rekey out after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
```

```

debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: rekey in after 134217728 blocks
debug1: Will attempt key: /home/smart/.ssh/id_rsa RSA
↳ SHA256:e/fioi65Tz2gyc473GqAAYvNcYKpMk6hh0aoYFsYJK4
debug1: Will attempt key: /home/smart/.ssh/id_ecdsa
debug1: Will attempt key: /home/smart/.ssh/id_ecdsa_sk
debug1: Will attempt key: /home/smart/.ssh/id_ed25519
debug1: Will attempt key: /home/smart/.ssh/id_ed25519_sk
debug1: Will attempt key: /home/smart/.ssh/id_xmss
debug1: Will attempt key: /home/smart/.ssh/id_dsa
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,sk-ssh-ed25519@
↳ openssh.com,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nis
↳ tp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,sk-ecdsa-sha2-nistp256
↳ @openssh.com,webauthn-sk-ecdsa-sha2-nistp256@openssh.com>
debug1: kex_input_ext_info: publickey-hostbound@openssh.com=<0>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering public key: /home/smart/.ssh/id_rsa RSA
↳ SHA256:e/fioi65Tz2gyc473GqAAYvNcYKpMk6hh0aoYFsYJK4
debug1: Authentications that can continue: publickey,password
debug1: Trying private key: /home/smart/.ssh/id_ecdsa
debug1: Trying private key: /home/smart/.ssh/id_ecdsa_sk
debug1: Trying private key: /home/smart/.ssh/id_ed25519
debug1: Trying private key: /home/smart/.ssh/id_ed25519_sk
debug1: Trying private key: /home/smart/.ssh/id_xmss
debug1: Trying private key: /home/smart/.ssh/id_dsa
debug1: Next authentication method: password
user@192.168.124.45's password:
debug1: Enabling compression at level 6.
Authenticated to 192.168.124.45 ([192.168.124.45]:22) using "password".
debug1: setting up multiplex master socket
debug1: channel 0: new mux listener
↳ [/tmp/ssh-master-socket-user@192.168.124.45:22] (inactive timeout: 0)
debug1: control_persist_detach: backgrounding master process

```

```

debug1: forking to background
debug1: Entering interactive session.
debug1: pledge: id
debug1: multiplexing control connection
debug1: channel 1: new mux-control [mux-control] (inactive timeout: 0)
debug1: channel 2: new session [client-session] (inactive timeout: 0)
debug1: client_input_global_request: rtype hostkeys-00@openssh.com
↳ want_reply 0
debug1: client_input_hostkeys: searching /home/smart/.ssh/known_hosts
↳ for 192.168.124.45 / (none)
debug1: client_input_hostkeys: searching /home/smart/.ssh/known_hosts2
↳ for 192.168.124.45 / (none)
debug1: client_input_hostkeys: hostkeys file
↳ /home/smart/.ssh/known_hosts2 does not exist
debug1: client_input_hostkeys: no new or deprecated keys from server
debug1: Sending environment.
debug1: channel 2: setting env LANG = "en_US.UTF-8"
debug1: mux_client_request_session: master session id: 2
Welcome to Ubuntu 22.10 (GNU/Linux 5.19.0-31-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

0 updates can be applied immediately.

Last login: Thu Feb 16 19:51:12 2023 from 192.168.124.62
~
user@ubuntu$
logout
debug1: client_input_channel_req: channel 2 rtype exit-status reply 0
debug1: client_input_channel_req: channel 2 rtype eow@openssh.com reply 0
debug1: channel 2: free: client-session, nchannels 3
debug1: channel 1: free: mux-control, nchannels 2
Shared connection to 192.168.124.45 closed.

```

## Выводы

В этой работе мы познакомились с дополнительными модулями iptables.

Гибкое комбинирование различных правил позволяет добиться максимально конкретного результата.



# Лабораторная работа 10. Ограничение количества соединений

**Цель работы:** Исследовать средства ограничения количества соединений к серверу с клиентских компьютеров.

## Введение

Linux, как и все операционные системы общего назначения, работает с максимальной эффективностью. Это означает, что, как правило, пока есть доступные ресурсы, приложения могут их запрашивать. С другой стороны, если какой-то пул ресурсов истощается, это может повлиять на работоспособность и скорость отклика всей системы.

Таким образом, даже если теоретически Linux не должен ограничивать использование ресурсов до пределов аппаратного обеспечения, на практике должен. Многие атаки типа "отказ в обслуживании" (DoS) работают пытаясь истощить целевые ресурсы. Чтобы избежать серьезных последствий, в любой современной операционной системе по умолчанию действуют политики использования ресурсов. Администратору может потребоваться настроить политики ограничения в соответствии с их вариантами использования. Кроме того, значения по умолчанию обычно подходят для общего использования.

Существует множество средств контроля безопасности, обеспечивающих стабильность и быстрое действие системы.

## 1. Файловые дескрипторы

Способ, которым Linux и другие операционные системы на основе POSIX взаимодействуют между процессами, называется межпроцессным взаимодействием или IPC. Одна из прелесть этой концепции заключается в том, что она применима к связи между процессами на

одном хосте или через сеть компьютеров. Это означает, что оба сценария имеют общую основу для базовых API.

Если у нас есть две программы на одном хосте, которые общаются друг с другом с помощью Sockets API (стандарт де-факто для потоковой передачи данных в POSIX), их преобразование для работы на разных серверах потребует минимальных изменений. Следовательно, ядро предоставляет конечные точки связи в аналогичных формах.

Стоит заметить, что Sockets IPC API, используемый в Linux TCP/IP-соединениях, использует файловые дескрипторы. Таким образом, количество открытых файловых дескрипторов является одним из первых ограничений, с которыми можно столкнуться. Это относится как к сокетам TCP, так и к UDP.

## 1.1 Ограничения файловых дескрипторов на уровне ядра

Значения уровня ядра применимы ко всей системе. Количество доступных дескрипторов содержится в `/proc/sys/fs/file-max`.

```
user@ubuntu$ cat /proc/sys/fs/file-max
9223372036854775807
```

Это огромное число используется по умолчанию во многих дистрибутивах. Чтобы изменить это ограничение, мы можем установить его на лету с помощью команды `sysctl`.

```
user@ubuntu$ sudo sysctl fs.file-max=65536
[sudo] password for user:
fs.file-max = 65536
```

Мы ограничили количество дескрипторов до первой перезагрузки. Чтобы сделать эти изменения постоянными, нужно добавить запись в файл `/etc/sysctl.conf`, где можно установить постоянные настройки, записав строку:

```
fs.file-max=65536 # Ограничение количества открытых дескрипторов (файлов)
```

Всякий раз, когда предел будет достигнут, система выдаст событие "Слишком много открытых файлов в системе". Текущее использование дескриптором можно увидеть следующим образом:

```
user@ubuntu$ cat /proc/sys/fs/file-nr
6592      0      65536
```

Здесь мы видим три числа:

- текущее число используемых файловых дескрипторов;
- выделенное, но неиспользуемое число (всегда 0);
- максимальное число (то же, что и `fs.file-max`)

Наряду с общесистемным ограничением, ядро Linux налагает ограничения файлового дескриптора на каждый процесс. Это настраивается с помощью параметра `fs.nr_open`. Значение по умолчанию – 1048576 (снова довольно высокое значение).

## 1.2 Ограничения дескрипторов на уровне пользователя

Фактические ограничения накладываются оболочкой (shell) на пользовательском уровне. Каждый экземпляр оболочки устанавливает гораздо более строгий предел – по умолчанию 1024 открытых файла.

Этот лимит более чем подходит для обычных пользователей. Однако для серверных приложений он, скорее всего, достаточно низкий. Например, большие серверы баз данных могут иметь тысячи файлов данных и открытых соединений.

Этими ограничениями можно управлять с помощью команды `ulimit` и сохранять их, редактируя файл `/etc/security/limits.conf`. Например, чтобы изменить ограничение процесса `Oracle` на 8192, нужно добавить в файл эту строку:

#<domain>	<type>	<item>	<value>
oracle	hard	nofile	8192

Ключевое слово **hard** означает, что непривилегированные пользователи не могут изменять ограничение в любой момент. Мягкое ограничение позволит пользователю без полномочий `root` использовать команду `ulimit`, чтобы изменить его для определенных случаев использования.

## 2. Процессы и потоки

Как и в случае с ограничением не количество файловых дескрипторов, существуют ограничения как ядра, так и пользовательского пространства на количество процессов и потоков. В серверных приложениях мы обычно назначаем соединения рабочим процессам (workers). Таким образом, их ограничения могут ограничивать количество соединений, которые они могут обрабатывать.

Для процессов ограничивающими параметрами являются:

- Пространство ядра: `kernel.pid_max`. По умолчанию 32767 и управляет общесистемным размером таблицы процессов.
- Пространство пользователя: `ulimit -u` или параметр `nproc` в `limit.conf`. Максимальное количество пользовательских процессов 15397.

И, для потоков:

- Пространство ядра: `kernel.threads-max`. Максимальное количество потоков, которые может создать системный вызов `fork`. Его можно уменьшить в реальном времени, когда таблица процессов достигает 1/8 оперативной памяти системы.
- Пространство пользователя: общая виртуальная память / (размер стека \* 1024 \* 1024). Размер стека контролируется с помощью `ulimit -s` или элемента `stack` в `limit.conf`.

## 3. Параметры сетевого стека

Существуют параметры ядра, которые могут косвенно влиять на количество TCP-соединений. TCP имеет довольно сложный конечный автомат и ядро должно отслеживать каждое состояние соединения (тайминги и переходы). Кроме управляющих таблиц TCP, можно рассмотреть настройки Netfilter, которые так же могут влиять на ограничение TCP-соединений.

Для Netfilter:

- `net.netfilter.nf_conntrack_max`: максимальное количество подключений для отслеживания.

- `nf_conntrack_tcp_timeout_*`: ограничивает тайм-аут для каждого состояния TCP-соединения (отправка/получение SYN, ожидание закрытия, прочие таймауты)

Для стека TCP:

- `net.core.netdev_max_backlog`: максимальное количество пакетов в очереди на стороне получения, когда интерфейс получает пакеты быстрее, чем ядро может их обработать
- `net.ipv4.ip_local_port_range`: виртуальный диапазон портов (порты, динамически выделяемые на клиентской стороне соединений TCP).
- `net.ipv4.somaxconn`: размер очереди установленных соединений ожидающих обработки `accept()`.
- `net.ipv4.tcp_fin_timeout`: время, в течение которого потерянное соединение будет ждать, прежде чем оно будет прервано (состояние `TIME_WAIT`)
- `net.ipv4.tcp_tw_reuse`: позволяет повторно использовать сокеты с ожиданием времени для новых подключений, экономит ресурсы при высоких скоростях создания и уничтожения соединений.
- `net.ipv4.tcp_max_orphans`: максимальное количество сокетов TCP, не прикрепленных к дескриптору файла.
- `net.ipv4.tcp_max_syn_backlog`: максимальное количество запомненных запросов на подключение (`SYN_RECV`), которые не получили подтверждения от подключающегося клиента.
- `net.ipv4.tcp_max_tw_buckets`: Максимальное число сокетов, находящихся в состоянии `TIME-WAIT` одновременно.

Некоторые другие полезные параметры можно найти по ссылкам:

- Малоизвестные настройки ([opennet.ru](http://opennet.ru))
- Сказ о `sysctl` ([habr.com](http://habr.com))

Значение по умолчанию для этих параметров подходит для многих приложений. Как и другие параметры ядра, их значения можно установить с помощью команды `sysctl`, а сохранить изменения можно с используя файл `/etc/sysctl.conf`.

## 4. Ограничения IP Tables

В предыдущих работах мы уже отмечали, что можем использовать ip tables для назначения лимитов соединений. Мы можем установить ограничения на основе исходных адресов, портов назначения и многих других параметров. При этом используются модуль `connlimit` (или более новый `hashlimit`) ip tables.

Например, чтобы ограничить SSH-подключения до трёх на один IP-адрес, можно использовать:

```
iptables -append INPUT -protocol tcp -syn -dport 22 -match connlimit -connlimit-above 3 -jump REJECT
```

Где `-syn` позволяет пропускать пакеты TCP с установленным флагом SYN и снятыми ACK,RST,FIN. Такие пакеты используются для запроса создания TCP-соединения. Очевидно, блокирование таких пакетов приведёт к невозможности создания входящих TCP-соединений.

На практике лучше избегать такой подход и по возможности использовать соответствующий функционал в приложении, чтобы не грузить файрволл в ядре stateful вычислениями.

## 5. Ограничения запросов на уровне приложений

Веб-сервер NGINX имеет различные модули, позволяющие контролировать конечный трафик на свои веб-сайты, веб-приложения и другие ресурсы. Одной из основных причин ограничения трафика или доступа является предотвращение злоупотреблений или атак определенных видов, таких как DoS-атаки (отказ в обслуживании).

Существует три основных способа ограничения использования или трафика в NGINX:

- Ограничение количества подключений (запросов).
- Ограничение скорости запросов.
- Ограничение пропускного канала.

Эти подходы к управлению трафиком, могут быть настроены для ограничения на основе определенного ключа. Наиболее распространенным ключом которых является IP-адрес клиента, но также поддерживаются другие переменные, такие как cookie-файл, сеанс, и многие другие.

## 5.1 Ограничение количества подключений (запросов)

Первое что нужно сделать, это определить зону общей памяти, в которой будут храниться метрики подключения для различных ключей. За это отвечает директива `limit_conn_zone`. Эта директива задаётся в контексте HTTP, и принимает два параметра – ключ и зону (в формате `zone_name:size`).

```
limit_conn_zone $binary_remote_addr zone=limitconnbyaddr:20m;
```

Чтобы установить код состояния ответа, который возвращается на отклоненные запросы, используется директива `limit_conn_status`, которая принимает в качестве параметра код состояния HTTP. Она задаётся в контексте HTTP, `server` и `location`.

```
limit_conn_status 429;
```

Чтобы ограничить количество подключений, используется директива `limit_conn`. Она задаёт используемую зону памяти и максимальное количество разрешенных подключений, как показано в следующем фрагменте конфигурации. Эта директива задаётся в контексте HTTP, `server` и `location`.

```
limit_conn limitconnbyaddr 50;
```

Полный конфигурационный файл будет выглядеть так (для одного клиента разрешается иметь только 1 подключение):

```
1 limit_conn_zone $binary_remote_addr zone=limitconnbyaddr:1m;
2 limit_conn_status 429;
3
4 server {
5     listen 80;
6     server_name localhost;
7
8     access_log /var/log/nginx/host.access.log main;
9
10    limit_conn limitconnbyaddr 1;
11
12    location / {
13        root /usr/share/nginx/html;
14        index index.html index.htm;
15    }
16 }
```

## Запуск Nginx

```
smart@thinkpad$ docker run -it --rm -v
↳ ./default.conf:/etc/nginx/conf.d/default.conf:ro -p 80:80 nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt
↳ to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching
↳ /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
↳ /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
↳ /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching
↳ /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching
↳ /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/02/18 18:26:54 [notice] 1#1: using the "epoll" event method
2023/02/18 18:26:54 [notice] 1#1: nginx/1.23.3
2023/02/18 18:26:54 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian
↳ 10.2.1-6)
2023/02/18 18:26:54 [notice] 1#1: OS: Linux 6.1.12-arch1-1
2023/02/18 18:26:54 [notice] 1#1: getrlimit(RLIMIT_NOFILE):
↳ 1073741816:1073741816
2023/02/18 18:26:54 [notice] 1#1: start worker processes
2023/02/18 18:26:54 [notice] 1#1: start worker process 29
2023/02/18 18:26:54 [notice] 1#1: start worker process 30
2023/02/18 18:26:54 [notice] 1#1: start worker process 31
2023/02/18 18:26:54 [notice] 1#1: start worker process 32
2023/02/18 18:26:54 [notice] 1#1: start worker process 33
2023/02/18 18:26:54 [notice] 1#1: start worker process 34
2023/02/18 18:26:54 [notice] 1#1: start worker process 35
2023/02/18 18:26:54 [notice] 1#1: start worker process 36
2023/02/18 18:26:54 [notice] 1#1: start worker process 37
2023/02/18 18:26:54 [notice] 1#1: start worker process 38
2023/02/18 18:26:54 [notice] 1#1: start worker process 39
```



```
2023/02/18 18:26:54 [notice] 1#1: start worker process 40
```

Для тестирования, запускаем curl в несколько потоков, и очень быстро обнаруживаем в логах nginx уведомление об отказе обслуживания очередного запроса из-за превышения количества разрешённых соединений.

```
2023/02/18 18:45:35 [error] 31#31: *3834 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:35 [error] 32#32: *3840 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:35 [error] 21#21: *3846 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:35 [error] 21#21: *3856 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:35 [error] 22#22: *3870 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:36 [error] 23#23: *3881 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:36 [error] 23#23: *3887 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:36 [error] 24#24: *3895 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:37 [error] 27#27: *3926 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
2023/02/18 18:45:37 [error] 26#26: *3944 limiting connections by zone
↳ "limitconnbyaddr", client: 172.17.0.1, server: localhost, request:
↳ "GET / HTTP/1.1", host: "localhost"
```

## 5.2 Ограничение скорости запросов

Ограничение скорости — это метод управления трафиком, используемый для ограничения количества HTTP-запросов, которые клиент может сделать за определенный период времени. Ограничения скорости рассчитываются в количестве запросов в секунду (или RPS).

Примером запроса является запрос GET для страницы входа в приложение или запрос POST для формы входа или POST для конечной точки API.

Существует множество причин для ограничения скорости запросов веб-приложениям или службам API, одна из которых связана с безопасностью: защита от неправомерных быстрых запросов.

Определения параметров ограничения скорости выполняется помощью директивы `limit_req_zone`. Обязательными параметрами являются:

- ключ для идентификации клиентов
- зона общей памяти, в которой будет храниться состояние ключа и частота обращения к URL-адресу с ограничением запросов
- предельная скорость запроса

Директива `limit_req_zone` определяется в контексте HTTP.

```
limit_req_zone $binary_remote_addr zone=limitreqsbyaddr:20m rate=10r/s;
```

Аналогично предыдущему примеру, установим код состояния ответа, который возвращается на отклоненные запросы, используя директиву `limit_req_status`, которая определяется в контексте HTTP, `server` и `location`.

```
limit_req_status 429;
```

Включить ограничение скорости запросов нужно используя директиву `limit_conn`. Она определяется в контексте HTTP, `server` и `location`. В качестве параметров она принимает зону памяти.

```
limit_req=limitreqsbyaddr;
```

В следующем примере конфигурации показано ограничение частоты запросов к API веб-приложения. Размер общей памяти составляет 1 МБ, а ограничение скорости запросов — 1 запрос в секунду.

```

1 limit_req_zone $binary_remote_addr zone=limitreqsbyaddr:1m rate=1r/s;
2 limit_req_status 429;
3
4 server {
5     listen 80;
6     server_name localhost;
7
8     access_log /var/log/nginx/host.access.log main;
9
10    limit_req zone=limitreqsbyaddr;
11
12    location / {
13        root /usr/share/nginx/html;
14        index index.html index.htm;
15    }
16 }

```

Запускаем контейнер с этим конфигурационным файлом

```

smart@thinkpad$ docker run -it --rm -v
↪ ./default2.conf:/etc/nginx/conf.d/default.conf:ro -p 80:80 nginx

```

Выполняем пару запросов curl с интервалом меньше 1 секунды. Запрос nginx отклонён с кодом 429.

```

smart@thinkpad$ curl localhost:80
<html>
<head><title>429 Too Many Requests</title></head>
<body>
<center><h1>429 Too Many Requests</h1></center>
<hr><center>nginx/1.23.3</center>
</body>
</html>

```

В логах nginx появляется такая запись.

```
2023/02/18 19:04:20 [error] 21#21: *21 limiting requests, excess: 0.777
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
2023/02/18 19:04:21 [error] 21#21: *22 limiting requests, excess: 0.665
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
2023/02/18 19:04:21 [error] 21#21: *23 limiting requests, excess: 0.557
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
2023/02/18 19:04:21 [error] 21#21: *24 limiting requests, excess: 0.443
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
2023/02/18 19:04:21 [error] 21#21: *25 limiting requests, excess: 0.330
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
2023/02/18 19:04:26 [error] 21#21: *27 limiting requests, excess: 0.735
↳ by zone "limitreqsbyaddr", client: 172.17.0.1, server: localhost,
↳ request: "GET / HTTP/1.1", host: "localhost"
```

### 5.3 Ограничение пропускного канала

Чтобы гарантировать, что пропускная способность приложения не расходуется на одного "толстого" клиента (с большим пропускным каналом), необходимо контролировать скорость загрузки и выгрузки для каждого клиента. Это обычная защита NGINX от DoS-атак со стороны злоумышленников, которые просто пытаются злоупотребить производительностью сайта.

Для ограничения пропускного канала в NGINX используется директива `limit_rate`, которая ограничивает скорость передачи ответа клиенту. Она определяется в контекстах HTTP, server, location и if блоках внутри location. По умолчанию указывает ограничение скорости в байтах в секунду, но можно использовать `m` для мегабайт или `g` для гигабайт.

```
limit_rate 20k;
```

С ней связана директива `limit_rate_after`, которая указывает что соединение не должно быть ограничено по скорости до тех пор, пока не будет передано указанное количество данных. Эта директива может быть задана в тех же контекстах, что и `limit_rate`.

```
limit_rate_after 500k;
```

В этом примере мы ограничиваем загрузку клиентом содержимого до максимальной скорости 5 килобайт в секунду.

```
1 server {
2     listen 80;
3     server_name localhost;
4
5     access_log /var/log/nginx/host.access.log main;
6
7     limit_rate 5k;
8     limit_rate_after 1k;
9
10    location / {
11        root /usr/share/nginx/html;
12        index index.html index.htm;
13    }
14 }
```

Запуск сервера

```
smart@thinkpad$ docker run -it --rm -v
↪ ./default3.conf:/etc/nginx/conf.d/default.conf:ro -p 80:80 nginx
```

Целевой документ крайне мал по объёму, но после нескольких попыток можно заметить замедление скорости передачи.

```
smart@thinkpad$ curl -o /dev/null -s -w %{time_total}\\n localhost
0.000882

smart@thinkpad$ curl -o /dev/null -s -w %{time_total}\\n localhost
0.000871

smart@thinkpad$ curl -o /dev/null -s -w %{time_total}\\n localhost
0.000566

smart@thinkpad$ curl -o /dev/null -s -w %{time_total}\\n localhost
0.000861
```

## Выводы

В этой работе мы рассмотрели различные механизмы ограничения скорости соединения, и обозначили в каких ситуациях это нужно.

С практической стороны, нет универсального решения, и каждый потенциальный вектор атаки должен встречать свой подход для защиты.

# Лабораторная работа 11. Сниффер заголовков сообщений протоколов уровней L2 и L3 модели OSI

**Цель работы:** Исследуйте возможности RAW-сокетов предоставляющих доступ к полям заголовков сообщений протоколов уровней L2 и L3 модели OSI.

## Введение

Исследование возможностей RAW-сокетов, предоставляющих доступ к полям заголовков сообщений протоколов уровней L2 и L3 модели OSI, является одной из ключевых тем в области сетевых технологий. RAW-сокеты представляют собой механизм, который позволяет разработчикам программного обеспечения взаимодействовать непосредственно с сетевым стеком операционной системы, минуя обычные сетевые протоколы высокого уровня.

В рамках модели OSI, уровень L2 (Data Link Layer) отвечает за передачу данных между соседними узлами сети, обеспечивая физическую адресацию (MAC-адреса) и управление доступом к среде передачи данных. На этом уровне работают протоколы Ethernet, Wi-Fi и другие. RAW-сокеты позволяют программам обрабатывать пакеты на уровне L2, получая доступ к полям заголовка Ethernet-кадра, например, исследовать и управлять MAC-адресами или управлять параметрами фреймов.

Уровень L3 (Network Layer) отвечает за маршрутизацию и доставку данных между сетями. Здесь работают протоколы IP, ICMP, IPv6 и другие. RAW-сокеты, предоставляющие доступ к L3-заголовкам, позволяют программам анализировать и модифицировать IP-адреса, проверять контрольные суммы и другие параметры протокола IP, а также работать с другими протоколами на уровне L3.

Использование RAW-сокетов может быть полезным во множестве сценариев, таких как

разработка сетевых приложений, отладка и тестирование сетевых протоколов, реализация сетевых утилит и многого другого. Однако следует отметить, что работа с RAW-сокетами требует определенных привилегий и может быть ограничена на некоторых платформах или в некоторых сетевых окружениях с целью обеспечения безопасности и предотвращения злоумышленнической деятельности.

## 1. Разработка приложения

Требования к приложению состоят в разработке и отладке консольное приложение, обладающего возможностями сниффера пакетов, используя технологию RAW-сокетов. Необходимо ориентироваться на сообщения протоколов ARP, ICMP, UDP, TCP. Для разработки был выбран язык разработки Rust с целью исследования его возможностей в данной области.

Листинг 9: Исходный код сниффера

```
1 use pdu::*;
2 use rawsock::open_best_library;
3
4 fn main() {
5     println!("Opening packet capturing library");
6     let lib = open_best_library().expect("Could not open any packet
7         ↪ capturing library");
8     println!("Library opened, version is {}", lib.version());
9     let interf_name = "wlp3s0"; //replace with whatever is available on your
10        ↪ platform
11     println!("Opening the {} interface", interf_name);
12     let mut interf = lib
13         .open_interface(&interf_name)
14         .expect("Could not open network interface");
15     println!("Interface opened, data link: {}", interf.data_link());
16
17     //receive some packets.
18     println!("Receiving 5 packets:");
19     for _ in 0..5 {
20         let packet = interf.receive().expect("Could not receive packet");
21         println!("Received packet: {:02x?}", packet.as_ref());
22
23         match EthernetPdu::new(&packet) {
24             Ok(ethernet_pdu) => {
25                 println!(
26                     "[ethernet] destination_address: {:x?}",
27                     ethernet_pdu.destination_address().as_ref()
28                 );
29             }
30         }
31     }
32 }
```



```

27     println!(
28         "[ethernet] source_address: {:x?}",
29         ethernet_pdu.source_address().as_ref()
30     );
31     println!(
32         "[ethernet] ethertype: 0x{:04x}",
33         ethernet_pdu.ethertype()
34     );
35     if let Some(vlan) = ethernet_pdu.vlan() {
36         println!(
37             "[ethernet] vlan: 0x{:04x}",
38             vlan
39         );
40     }
41     match ethernet_pdu.inner() {
42         Ok(Ethernet::Arp(arp_pdu)) => {
43             println!(
44                 "[ARP] operation code: 0x{:02x}",
45                 arp_pdu.opcode()
46             );
47             println!(
48                 "[ARP] Sender hardware address: {:x?}",
49                 arp_pdu.sender_hardware_address().as_ref()
50             );
51             println!(
52                 "[ARP] Sender protocol address: {:x?}",
53                 arp_pdu.sender_protocol_address().as_ref()
54             );
55             println!(
56                 "[ARP] Target hardware address: {:x?}",
57                 arp_pdu.target_hardware_address().as_ref()
58             );
59             println!(
60                 "[ARP] Target protocol address: {:x?}",
61                 arp_pdu.target_protocol_address().as_ref()
62             );
63         }
64         Ok(Ethernet::Ipv4(ipv4_pdu)) => {
65             println!(
66                 "[ipv4] source_address: {:x?}",
67                 ipv4_pdu.source_address().as_ref()
68             );
69             println!(
70                 "[ipv4] destination_address: {:x?}",
71                 ipv4_pdu.destination_address().as_ref()

```

```

72     );
73     println!(
74         "[ipv4] protocol: 0x{:02x}",
75         ipv4_pdu.protocol()
76     );
77     match ipv4_pdu.inner() {
78         Ok(Ipv4::Tcp(tcp_pdu)) => {
79             println!(
80                 "[TCP] Source port: {:x?}",
81                 tcp_pdu.source_port()
82             );
83             println!(
84                 "[TCP] Destination port : {:x?}",
85                 tcp_pdu.destination_port()
86             );
87             println!(
88                 "[TCP] Sequence number: {:x?}",
89                 tcp_pdu.sequence_number()
90             );
91         }
92         Ok(Ipv4::Udp(udp_pdu)) => {
93             println!(
94                 "[UDP] Source port: {:x?}",
95                 udp_pdu.source_port()
96             );
97             println!(
98                 "[UDP] Destination port : {:x?}",
99                 udp_pdu.destination_port()
100            );
101            println!(
102                "[UDP] Length: {:x?}",
103                udp_pdu.length()
104            );
105        }
106        Ok(Ipv4::Icmp(icmp_pdu)) => {
107            println!(
108                "[ICMP] Message code: 0x{:02x?}",
109                icmp_pdu.message_code()
110            );
111            println!(
112                "[ICMP] Message type : 0x{:02x?}",
113                icmp_pdu.message_type()
114            );
115            println!(
116                "[ICMP] Checksum: {:x?}",

```

```

117         icmp_pdu.checksum()
118     );
119 }
120 Ok(other) => {
121     println!(
122         "Other (unexpected) packet {:?}",
123         other
124     );
125 }
126 Err(e) => {
127     panic!("Ipv4::inner() parser failure: {:?}",
128         ↪ e);
129 }
130 }
131 Ok(Ethernet::Ipv6(ipv6_pdu)) => {
132     println!(
133         "[ipv6] source_address: {:x?}",
134         ipv6_pdu.source_address().as_ref()
135     );
136     println!(
137         "[ipv6] destination_address: {:x?}",
138         ipv6_pdu.destination_address().as_ref()
139     );
140     println!(
141         "[ipv6] protocol: 0x{:02x}",
142         ipv6_pdu.computed_protocol()
143     );
144     match ipv6_pdu.inner() {
145         Ok(Ipv6::Tcp(tcp_pdu)) => {
146             println!(
147                 "[TCP] Source port: {:x?}",
148                 tcp_pdu.source_port()
149             );
150             println!(
151                 "[TCP] Destination port : {:x?}",
152                 tcp_pdu.destination_port()
153             );
154             println!(
155                 "[TCP] Sequence number: {:x?}",
156                 tcp_pdu.sequence_number()
157             );
158         }
159         Ok(Ipv6::Udp(udp_pdu)) => {
160             println!(

```

```

161         "[UDP] Source port: {:x?}",
162         udp_pdu.source_port()
163     );
164     println!(
165         "[UDP] Destination port : {:x?}",
166         udp_pdu.destination_port()
167     );
168     println!(
169         "[UDP] Length: {:x?}",
170         udp_pdu.length()
171     );
172 }
173 Ok(Ipv6::Icmp(icmp_pdu)) => {
174     println!(
175         "[ICMP] Message code: 0x{:02x?}",
176         icmp_pdu.message_code()
177     );
178     println!(
179         "[ICMP] Message type : 0x{:02x?}",
180         icmp_pdu.message_type()
181     );
182     println!(
183         "[ICMP] Checksum: {:x?}",
184         icmp_pdu.checksum()
185     );
186 }
187 Ok(other) => {
188     println!(
189         "Other (unexpected) packet {:?}" ,
190         other
191     );
192 }
193 Err(e) => {
194     panic!("Ipv4::inner() parser failure: {:?}" ,
195         ↪ e);
196 }
197 }
198 Ok(other) => {
199     println!(
200         "Other (unexpected) protocol {:?}" ,
201         other
202     );
203 }
204 Err(e) => {

```

```

205         panic!("EthernetPdu::inner() parser failure: {:?}",
           ↪ e);
206     }
207 }
208 }
209 Err(e) => {
210     panic!("EthernetPdu::new() parser failure: {:?}", e);
211 }
212 }
213 }
214 }

```

## 2. Демонстрация работы

Демонстрация возможности перехвата сообщений отдельных протоколов с помощью разработанного приложения представлена в следующем логе. Она показывает не все возможности приложения (захватываются только 5 случайных пакетов).

Отдельно стоит отметить, что доступ к RAW сокетам требует уровень доступа суперпользователя, таким образом, сниффер должен запускаться от root.

Листинг 10: Лог работы сниффера: захват 5 случайных пакетов

```

1 Opening packet capturing library
2 Library opened, version is  pcap libpcap version 1.10.4 (with TPACKET_V3)
3 Opening the wlp3s0 interface
4 Interface opened, data link: ethernet
5 Receiving 5 packets:
6 Received packet: [34, ce, 00, 37, d9, 03, 14, 5a, fc, 0d, 56, 2d, 86, dd,
  ↪ 60, 00, 00, 00, 00, 20, 3a, ff, fe, 80, 00, 00, 00, 00, 00, 00, 27,
  ↪ 06, a7, 1b, 9b, ab, 47, a0, fe, 80, 00, 00, 00, 00, 00, 00, 36, ce,
  ↪ 00, ff, fe, 37, d9, 03, 87, 00, 46, 0c, 00, 00, 00, 00, fe, 80, 00,
  ↪ 00, 00, 00, 00, 00, 36, ce, 00, ff, fe, 37, d9, 03, 01, 01, 14, 5a, fc
  ↪ , 0d, 56, 2d]
7 [ethernet] destination_address: [34, ce, 0, 37, d9, 3]
8 [ethernet] source_address: [14, 5a, fc, d, 56, 2d]
9 [ethernet] ethertype: 0x86dd
10 [ipv6] source_address: [fe, 80, 0, 0, 0, 0, 0, 0, 27, 6, a7, 1b, 9b, ab, 47,
  ↪ a0]
11 [ipv6] destination_address: [fe, 80, 0, 0, 0, 0, 0, 0, 36, ce, 0, ff, fe,
  ↪ 37, d9, 3]
12 [ipv6] protocol: 0x3a
13 [ICMP] Message code: 0x00

```

```

14 [ICMP] Message type : 0x87
15 [ICMP] Checksum: 460c
16 Received packet: [14, 5a, fc, 0d, 56, 2d, 34, ce, 00, 37, d9, 03, 86, dd,
    ↪ 60, 00, 00, 00, 00, 18, 3a, ff, fe, 80, 00, 00, 00, 00, 00, 00, 36, ce
    ↪ , 00, ff, fe, 37, d9, 03, fe, 80, 00, 00, 00, 00, 00, 00, 27, 06, a7,
    ↪ 1b, 9b, ab, 47, a0, 88, 00, ec, a9, c0, 00, 00, 00, fe, 80, 00, 00,
    ↪ 00, 00, 00, 00, 36, ce, 00, ff, fe, 37, d9, 03]
17 [ethernet] destination_address: [14, 5a, fc, d, 56, 2d]
18 [ethernet] source_address: [34, ce, 0, 37, d9, 3]
19 [ethernet] ethertype: 0x86dd
20 [ipv6] source_address: [fe, 80, 0, 0, 0, 0, 0, 0, 36, ce, 0, ff, fe, 37, d9,
    ↪ 3]
21 [ipv6] destination_address: [fe, 80, 0, 0, 0, 0, 0, 0, 27, 6, a7, 1b, 9b, ab
    ↪ , 47, a0]
22 [ipv6] protocol: 0x3a
23 [ICMP] Message code: 0x00
24 [ICMP] Message type : 0x88
25 [ICMP] Checksum: eca9
26 Received packet: [34, ce, 00, 37, d9, 03, 14, 5a, fc, 0d, 56, 2d, 08, 00,
    ↪ 45, 00, 00, 28, e1, cc, 40, 00, 40, 06, 32, f9, c0, a8, 7c, 3e, 14, b9
    ↪ , d4, 6a, a7, 34, 01, bb, a4, 9c, c7, 08, 0d, 19, bc, 06, 50, 10, 04,
    ↪ f3, a7, 22, 00, 00]
27 [ethernet] destination_address: [34, ce, 0, 37, d9, 3]
28 [ethernet] source_address: [14, 5a, fc, d, 56, 2d]
29 [ethernet] ethertype: 0x0800
30 [ipv4] source_address: [c0, a8, 7c, 3e]
31 [ipv4] destination_address: [14, b9, d4, 6a]
32 [ipv4] protocol: 0x06
33 [TCP] Source port: a734
34 [TCP] Destination port : 1bb
35 [TCP] Sequence number: a49cc708
36 Received packet: [14, 5a, fc, 0d, 56, 2d, 34, ce, 00, 37, d9, 03, 08, 00,
    ↪ 45, b8, 00, 28, 09, ab, 40, 00, 6e, 06, dc, 62, 14, b9, d4, 6a, c0, a8
    ↪ , 7c, 3e, 01, bb, a7, 34, 0d, 19, bc, 06, a4, 9c, c7, 09, 50, 10, 40,
    ↪ 02, 6c, 12, 00, 00]
37 [ethernet] destination_address: [14, 5a, fc, d, 56, 2d]
38 [ethernet] source_address: [34, ce, 0, 37, d9, 3]
39 [ethernet] ethertype: 0x0800
40 [ipv4] source_address: [14, b9, d4, 6a]
41 [ipv4] destination_address: [c0, a8, 7c, 3e]
42 [ipv4] protocol: 0x06
43 [TCP] Source port: 1bb
44 [TCP] Destination port : a734
45 [TCP] Sequence number: d19bc06
46 Received packet: [34, ce, 00, 37, d9, 03, 14, 5a, fc, 0d, 56, 2d, 08, 00,

```

```

    ↪ 45, 00, 00, ed, da, a6, 40, 00, 40, 06, e5, b0, c0, a8, 7c, 3e, 95, 9a
    ↪ , a7, 32, 9d, 52, 01, bb, fd, d6, c4, e6, 3c, b4, 99, 3a, 80, 18, 0c,
    ↪ 0e, 53, 79, 00, 00, 01, 01, 08, 0a, 5f, 5f, 03, e3, 2a, e1, 80, a6, e1
    ↪ , 99, c3, 6e, c9, 98, fe, 6a, 2f, ce, 80, 5b, dd, 94, cf, 6f, e4, 46,
    ↪ 50, 98, 24, b9, 28, 48, 2b, a6, ab, 40, e0, 92, f4, d9, 86, eb, b4, b9
    ↪ , 6d, bd, 18, 28, 40, fa, 46, e2, 20, ef, 09, 54, b9, c0, cd, 54, 98,
    ↪ cf, c0, f9, 35, e5, 84, a1, f2, 7e, 9c, 27, 92, 9b, a3, 81, e4, 9d,
    ↪ 51, cb, 5d, 1a, b8, e7, d2, 1f, 0a, bf, 19, d9, c1, 90, d6, db, 38, 9a
    ↪ , b2, 58, 28, 88, 61, cb, 48, e0, da, f6, 84, 68, 92, 03, 6f, 09, 25,
    ↪ 6c, 89, f9, b7, 5a, a0, 55, a2, 5c, fe, eb, 36, 22, da, 8e, ea, 7d, e2
    ↪ , e4, 3b, 7c, 02, dc, d5, a4, cd, 88, f4, 2b, 82, 82, a2, 11, a7, 16,
    ↪ f0, ea, 41, 7d, b1, 15, 91, 1f, 6c, fb, d6, a1, cf, 91, 8f, be, 14, b9
    ↪ , bb, d0, 56, 3d, 9c, 34, 83, 2b, 8e, 70, bf, fd, b4, 5a, 34, e3, 11,
    ↪ d0, 5b, a9, 1e, e6, f4, e2, 78, 66, d9]
47 [ethernet] destination_address: [34, ce, 0, 37, d9, 3]
48 [ethernet] source_address: [14, 5a, fc, d, 56, 2d]
49 [ethernet] ethertype: 0x0800
50 [ipv4] source_address: [c0, a8, 7c, 3e]
51 [ipv4] destination_address: [95, 9a, a7, 32]
52 [ipv4] protocol: 0x06
53 [TCP] Source port: 9d52
54 [TCP] Destination port : 1bb
55 [TCP] Sequence number: fdd6c4e6

```

Мы отказались от фильтрации протоколов и пакетов через параметры запускаемого sniffера и разбираем все пакеты, которые удаётся обнаружить на сетевом интерфейсе.

## Выводы

В рамках этой работы мы ставили задачу исследования возможностей предоставления доступа к полям заголовков сообщений через RAW-сокеты.

Если сопоставьте наши результаты с результатами работы других общеизвестных sniff-еров, можно отметить что разница в значительной части связана с интерфейсом, т.к. нижележащие библиотеки и системные вызовы используются одни и те же.

Касательно языка Rust можно отметить что его механика `pattern matching` крайне удобна для подобных задач. По мере накопления опыта в этом языке, можно перейти от процедурного подхода в коде sniffера, к чему-то более поддерживаемому.

В целом, исследование возможностей RAW-сокеты на уровнях L2 и L3 модели OSI представляет собой интересную и важную область, которая позволяет разработчикам более гибко и

эффективно управлять сетевыми операциями на низком уровне и создавать инновационные решения в сетевых технологиях.