Чёрт в табакерке: инструмент для диагностики сети на базе ОС FreeBSD

Автор: Залата Руслан Николаевич

tag: Космотекст

Изложенная в этой статье идея и инструкция по воплощению инструмента диагностики сетевых проблем, для многих сисадминов старой закалки может показаться банальной, но я уверен, что молодое поколение инженеров взросших на сертифицированных и дорогостоящих решениях от именитых производителей (Cisco Systems, Fluke Networks, etc) вряд ли даже догадывается о том, какой мощный инструмент можно получить от операционной системы FreeBSD прямо из коробки. Достаточно установить её на компактное

устройство с двумя Ethernet интерфейсами и выполнить ряд манипуляций по настройке вполне стандартных вещей.

Идея создания «stand-alone» инструмента для удаленной диагностики Ethernet сетей не покидала мою голову с начала 2000-х годов когда я еще трудился у интернет-провайдеров местного розлива и постоянно сталкивался с необходимостью анализа сетевого трафика для устранения различных проблем. В те времена мы очень часто использовали стандартное ПК-шное железо с установленной на него ОС FreeBSD для сетевых маршрутизаторов и серверов, так как на что-то более-менее серьезное просто не было средств, такие уж были времена. С тех пор я глубоко проникся сетевыми возможностями предоставляемыми этой операционной системой, как говорят, прямо из коробки — маршрутизатор и сетевой экран поднимаются на ней за считанные минуты на стандартном железе, которого хватало на долгие годы. Да что тут говорить, у меня на фирме уже более 10-ти лет маршрутизатором трудится Фря на старом HP-шном 1U сервере и даже не думает сдаваться.

Написать эту статью меня побудила нужда. Я уже давно не являюсь сетевым инженером и порядком отстал от современных тенденций. Но один из моих заказчиков, для которого мы выполняли разработку программно-аппаратного комплекса и сейчас осуществляем его сопровождение, эксплуатирует большой парк разношерстных IP-домофонов и постоянно сталкивается с неразрешимыми проблемами связанными с инфраструктурой. Эти домофоны обычно подключаются к Сети через сторонних интернет-провайдеров, до сервисных служб которых в последнее время достучаться очень сложно. Даже если удается пообщаться с инженером на стороне провайдера, то объяснить ему суть проблемы, и тем более доказать, что причина находится на его стороне, не имея каких-то технических фактов, — просто не реально. С другой стороны, огромное количество дешевых IP-домофонов китайского производства не предоставляют возможности выяснить, что и как там внутри происходит, какие специфические опции SIP протокола они используют и правильно ли используют. Существует масса проблем несовместимости по SIP и нерабочих механизмов «NAT traversal», а каких либо способов их диагностировать сами изделия не предоставляют. В общем, есть острая необходимость «подслушать» трафик создаваемый таким устройством и проанализировав его понять «кто виноват и что делать».

Link: https://www.fabmicro.ru/pub/articles/Daemon-in-the-Box-Sniffing-on-FreeBSD.pdf

СОДЕРЖАНИЕ

1. Сниффер как способ решения проблем с сетью

Одним из традиционных способов «подсмотреть» сетевой трафик — это настроить порт Ethernet коммутатора (или отдельный VLAN), трафик которого требуется мониторить, в специальный режим, при котором весь трафик проходящий по интересующему порту (или VLAN-у) будет полностью копироваться в другой, свободный порт. В терминах Cisco это называется SPAN. К этом свободному порту подключается анализирующее устройство, например Fluke OptiView, или ПК с запущенным на нём ПО WireShark для визуализации и непосредственно анализа трафика. Недостатком такого решения может быть следующее: во-первых оконечный коммутатор должен поддерживать фичу SPAN, то есть это коммутатор достаточно продвинутый. Во-вторых, требуется полный (админский) доступ к коммутатору, что может быть просто невозможно, если мы имеем дело с сетью провайдера. В третьих, нам всё еще требуется отдельный ноутбук, ПК или дорогостоящий девайс (Fluke) для того, чтобы непосредственно мониторить и записывать трафик. В четвертых, скорее всего нам придется физически присутствовать в точке съема трафика, что может быть не всегда возможным, да и просто неудобно — настоящий сетевой инженер должен сидеть в мягком рабочем кресле и пить кофе, а не ползать по грязным техплощадкам. Бесспорно, админы крупных компаний могут мониторить трафик в любой точке сети, их «железо» это позволяет, но до них нам не достучаться. И тут есть еще один очень немаловажный момент - в процессе мониторинга, одним неловким движением руки можно легко «окирпичить» большой сегмент сети (ошибка в конфиге, или коммутатор может просто не справится с нагрузкой) и именно по этой причине сетевые инженеры очень неохотно прибегают к таким средствам диагностики — трогать руками рабочее коммуникационное оборудование нельзя. Что же делать ? И тут мы плавно переходим к сути моей идеи.

Суть идеи, как многие уже догадались, сводится к следующему. Берем стандартный ПК с двумя Ethernet интерфейсами (каким образом у ПК появятся два Ethernet-а пока оставим за кадром) и делаем так, чтобы пакеты поступающие из одного сетевого интерфейса тут же попадали в другой, и наоборот. Такой фокус называется «Ethernet Bridging» (или просто «мост»). С точки зрения сетевого оборудования (L2 коммутатора), такой мост полностью прозрачен и почти незаметен - не вычисляя задержек обнаружить наличие моста просто невозможно. Но не в этом суть, а в том, что транслируя пакеты между сетевыми интерфейсами мы попутно можем их программно *анализировать*, *выбрасывать*, *изменять по своему усмотрению* и просто *записывать в файл* стандартного формата для дальнейшего анализа в спокойной обстановке за кружкой кофе, никому при этом не мешая. Для управления таким устройством мы можем использовать SSH через L2TP или PPTP туннель, который автоматически подымается операционной системой при загрузке. То есть получается полностью автономное, удаленно управляемое устройство для мониторинга сетевого трафика. Установка устройства мониторинга очень проста — подключить два UTP патч-корда и подать питание, с чем легко справятся техники из монтажной бригады.

Установив такое устройство, назовем его «сниффер» (от анг. sniff — нюхать), в разрез между двумя Ethernet коммутаторами мы получаем возможность анализировать весь проходящий между ними трафик и таким образом диагностировать огромный спектр сетевых проблем — от сетевого шторма, до неверно настроенных трансляторов сетевых адресов (NAT), проблем с маршрутизацией и проблем с протоколами более высокого уровня, как-то например SIP. В некоторых случаях нам даже удастся подглядывать в пользовательский трафик, но это уже на усмотрение моральных принципов сетевого инженера. Мы даже сможем выявлять проблемы безопасности и попыток вторжения!

Очевидно, что вычислительных ресурсов ПК на котором работает сниффер может быть не достаточно для того, чтобы анализировать большой поток трафика, поэтому втыкать такой сниффер в разрез магистральной сети я рекомендовать не возьмусь. ;) Хотя, если хорошо подготовиться, взять машинку подурнее, да с оптическими интерфейсами…

Короче, наш сниффер — это скорее инструмент для «бедного инженера» сталкивающегося с проблемами диагностирования сетевых неурядиц, расположенных ближе к абоненту, чем для сетевика респектабельной магистральной компании, которая может не задумываясь отвесить от 50 до 100 кило бакинских за какой нибудь Fluke OptiView XG с двумя 10GigabitEthernet. На eBay, кстати, такой продается за $4000, что тоже немалые деньги, плюс масса нюансов с доставкой в Россию.

Все что нам требуется для реализации идеи — это обычный x86 совместимый ПК с двумя сетевыми (GigabitEthernet) интерфейсами. До недавних времен найти такой ПК кроме как в серверном исполнении было не просто. Да, мы можем подключить к ноутбуку дополнительную сетевую карту через USB, но опыт показал, что это 1) весьма не надежно, и 2) достаточно громоздко и неудобно для долгосрочного мониторинга. Часто случается так, что проблема проявляется спорадически. Дожидаться пока проблема себя проявит, сидя несколько дней где-нибудь в подвале МКД с ноутбуком — «не айс». Еще хуже, если проблему надо мониторить у абонента в квартире или частном доме. Короче, нам нужно компактное и автономное решение. И это решение пришло с появлением в свободной продаже компактных ПК на базе процессоров Intel Atom с двумя и более Ethernet портами.

Где-то с пол года назад, один мой коллега, сетевой инженер у небольшого местного провайдера, скинул мне ссылку на интересное устройство, продаваемое на известном китайском маркетплейсе. Это был одноплатный ПК на базе Intel Atom J1190 (2.4 ГГц), выполненный в форм-факторе nano-ITX, производства малоизвестной китайской фирмы XCY. Данный ПК размерами 135x129x40mm был оснащен 8 GB оперативной памяти DDR4, накопителем SSD емкостью 128 GB, но самое интересное — у него целых четыре 2.5 Gbit Ethernet порта. И продавалось все это богатство за каких-то 8000 руб. Многие одноплатные ПК на базе ARM сейчас стоят дороже!

Рис.1. Мини-ПК от XCY на базе Intel Celeron J1900 с четырьмя портами 2.5GBit Ethernet.

Увидев этот мини-ПК я тут же вспомнил про идею сниффера и при удобном случае изложил её своему заказчику, эксплуатирующему сеть IP-домофонов. Поначалу идею не восприняли, точнее, её просто не поняли и не смогли оценить. Но прошло время, число нерешенных сетевых проблем накапливалось, число жалоб абонентов росло, и при случае я

задвинул эту идею еще раз. Так удачно получилось, что в компании заказчика появился квалифицированный специалист, который сразу понял как эту идею можно применить для решения их проблем и процесс завертелся. Приобрести мини-ПК от XCY не удалось в виду того, что компания не может вести закупки на китайских маркетплесах, но новый коллега быстро подобрал аналогичное устройство продаваемое в России. Это был mini-ПК Horizon N5 сингапурского дизайн-центра, который в России продается под брендом Rambica.

Рис.2. Мини-ПК Horizon N5 DN5122P [PCMI-0105], процессор Intel Celeron N5105 (4 ядра по 2.0 ГГц), 12 ГБ DDR4, 256 ГБ SSD, 2x1GbitEthernet, 3xUSB и 3xHDMI.

Rambica Horizon N5 DN5122P [PCMI-0105] оснащен немного другим процессором, а именно Intel Celeron N5105 (4 ядра по 2.0 ГГц). Устройство имеет 12 GB DDR4 RAM и накопитель SSD емкостью 256 GB. В отличии от XCY у этого Horizon N5 всего два 1 GbitEthernet порта, но этого более чем достаточно для создания на его базе сниффера для небольшой сети. А еще у него имеется встроенный WiFi, три порта HDMI, три порта USB 3.0 и стильная RGB подсветочка. ;-) Все это в габаритах 47x88x88мм и за 17 000 руб. При таких характеристиках и цене это серьезный удар по рынку одноплатников во главе с RaspberryPI 5.

Устройство было оперативно приобретено и передано мне для установки и настройки FreeBSD под поставленную задачу. В этой статье я хочу поделиться с читателем процессом установки, настройки и эксплуатации данного устройства во всех его деталях.

Итак, приступим к техническим подробностям реализации идеи сниффера.

2. Установка ОС FreeBSD и базовая настройка

Для установки операционной системы FreeBSD, настройке и тестирования сниффера нам потребуются следующие дополнительные средства:

1. Рабочий ПК с ОС Windows/Linux/FreeBSD или любой другой современной ОС с 3 ГБ свободного дискового пространства. На этом ПК должно быть средство позволяющее подключаться к другим машинам по SSH (ssh, putty.exe).
2. Накопитель USB Flash емкостью не менее 2 GB.
3. Монитор/телевизор с HDMI интерфейсом.
4. Клавиатура USB 2.0.
5. Два подготовленных UTP патч-корда.
6. Доступ к сети WiFi (пароль на доступ) с выходом в Интернет.

2.1. Скачиваем ISO образ

Первым делом нам потребуется скачать загрузочный образ системы с инсталлятором на борту. На сайте проекта FreeBSD представлено несколько различных версий этой операционной системы. На момент написания данной статьи имеется три различные ветки ОС FreeBSD: текущая в разработке **15.0-CURRENT**, устаревшая но всё еще поддерживаемая **13.5-RELEASE** и стабильная **14.3-RELEASE**. Скачивать будем стабильную версию 14.3-RELEASE. Для каждой версии ОС предложено несколько вариантов образов разного объема и назначения. В том числе:

- **dvd1** — содержит загрузочную систему предназначенную для инсталляции с DVD или USB, а также набор уже собранных бинарных файлов для графического интерфейса. Данный вариант самый объемный и предназначен для установки в качестве рабочего места на ПК пользователя.
- **disc1** — то же самое, только без собранных пакетов. Также вариант для рабочего места.
- **bootonly** — не содержит инсталлятора, но может быть загружена с CD-ROM или USB как временная рабочая ОС.
- **memstick** — оптимизирован для развертывания на USB носитель, содержит минимум достаточный для выполнения инсталляции.
- **mini-memctick** — то же самое, что memstick, но без инсталлятора. Больше подходит как средство восстановления после сбоя.
- Также на сайте представлены готовые образы для виртуальных машин арендованных у Amazon AWS EC2. Это всё нас сейчас не интересует.

А интересует нас минимальный образ с инсталлятором, то есть **memstick**. Переходим на страницу с образами и видим, что ОС FreeBSD доступна для нескольких различных платформ, в том числе для **amd64**, **arm64** (в нескольких вариантах), **i386**, **PowerPC**, **PowerPC64** (также в нескольких вариантах) и **RISCV64**. Для каждой из поддерживаемых платформ образ представлен в обычном (**.img**) и упакованном (**.img.xz**) формате.

В данном случае нас интересует платформа **amd64**, а значит качаем файл FreeBSD-14.3-RELEASE-amd64-memstick.img.xz который занимает примерно 800 МБ.

2.2. Настройка опций BIOS

Пока загрузочный образ скачивается, мы должны подключить к нашему Mini-ПК USB клавиатуру, HDMI монитор, подать питание, нажать кнопку Power и войти в BIOS чтобы изменить ряд настоек. Сразу хочу заметить, что по неизвестной мне причине проводные клавиатуры стандарта USB 1.0 с BIOS-ом данного мини-ПК не работают, а вот с беспроводной Logitech USB 2.0 всё получилось.

Действовать, как обычно, требуется оперативно — сразу после включения питания (что индицируется веселой иллюминацией RGB подсветки), необходимо многократно нажимать клавишу DEL еще до появления изображения на экране монитора. Если упустить момент, то устройство загрузит ранее установленную на SSD накопителе ОС неизвестного происхождения с очень странным и непонятным интерфейсом пользователя. ;)

Если же Вы были достаточно оперативны, то перед Вами откроется текстовое меню привычного AMI BIOS.

В меню BIOS необходимо изменить следующие настройки:

1. Установить последовательность загрузки: первым с USB: ***Boot → Hard Disk Drives: USB***.

2. Установить режим в который переходит устройство после сбоя (пропадания и восстановления) питания. Необходимо установить режим **S0**, чтобы устройство всегда включалось и загружалось сразу после подачи питания, не дожидаясь нажатия клавиши Power: ***Boot → State After G3: S0 State***.

После изменения настроек необходимо выйти из меню BIOS с сохранением внесенных изменений, то есть выбрать «**Save Settings and Exit**».

2.3. Развертывание образа на USB Flash

После скачивания образа его необходимо распаковать командой **xz**. Для записи распакованного образа Вам потребуется около 2 ГБ свободного дискового пространства.

```
# xz -d FreeBSD-14.3-RELEASE-amd64-memstick.img.xz
```

На странице сайта FreeBSD с загрузками приводится следующая команда **dd** с параметрами для записи образа на USB носитель:

```
# dd if=FreeBSD-14.3-RELEASE-amd64-memstick.img of=/dev/da0 bs=1m conv=sync
status=progress
```

Записать образ можно и не распаковывая, дабы зря не тратить дисковое пространство, следующей командой:

```
# zx -d -c FreeBSD-14.3-RELEASE-amd64-memstick.img.xz | dd of=/dev/da0 bs=1m
conv=sync status=progress
```

Эти команды необходимо выполнять от пользователя **root** (либо воспользоваться утилитой **sudo**). Команды одинаково хорошо работают как в ОС Linux, так и в ОС FreeBSD.

Единственное, что необходимо сделать при работе в Linux — поменять имя файла-устройства USB носителя на **/dev/sdb** (или как оно у Вас определилось, см в **dmesg**).

Однако, команды **dd** нет в ОС Windows. В последней придется предварительно скачать и установить утилиту **Rufus** или **USBImager** и следовать соответствующим инструкциям. Кстати, распаковать образ под ОС Windows можно утилитой **winrar.exe**, она поддерживает формат **xz**.

2.4. Загрузка с USB Flash и установка системы на SSD

Вставьте USB носитель в любой из USB разъемов, выключите и включите устройство. Если предварительные операции прошли удачно (BIOS настроен на загрузку с USB и образ успешно записан на USB Flash носитель), то в течении пары секунд Вам представится загрузчик от ОС FreeBSD который предложит несколько вариантов действий:

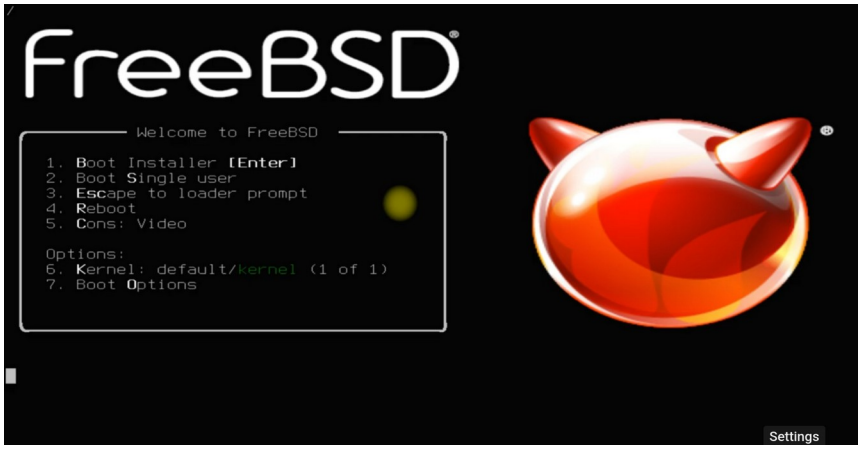

Рис.3. FreeBSD bootloader с образа memstick.img

В этот момент можно просто ничего не делать, а подождать 10 сек и активное (первое) меню «Boot Installer» будет выбрано автоматически. Либо нажать Enter чтобы ускорить выбор. Система продолжит загрузку, по экрану побегут строки текста от ядра ОС информирующие об обнаруженных устройствах, загруженных драйверах и их параметрах. В конечном счете Вам представится приглашение от FreeBSD Installer на голубом фоне:

Рис.4. Приглашение от FreeBSD Intaller-а.

Инсталлер предлагает три различных варианта действий, выбираем **[Install]**. Я не буду детально расписывать все шаги установки ОС FreeBSD и показывать скриншоты, в сети предостаточно статей на эту тему, а на Ютубе - десятки обучающих видео с пошаговой инструкцией. Укажу лишь последовательность действий и некоторые моменты с ними связанные:

1. Сначала Инсталлер предложит выбрать раскладку клавиатуры. В терминах FreeBSD это называется «Keymap». Можно отлистать на **Russian** и выбрать эту раскладку, но можно оставить и дефолтную **US** — это мало на что влияет в контексте нашей задачи.

2. Далее Инсталлер попросит указать имя машины. В среде Unix принято давать машинам (хостам) романтичные имена. В нашем случае предлагаю обозвать хост словом отражающем его будущую гнусную и мерзкую работу: **sniffer**.

3. Следующим шагом Инсталлер попросит выбрать набор дистрибутивов (набор стандартных пакетов). Отключаем всё, то касается отладки (***-dbg**), включаем **lib32** и двигаемся далее.

4. Далее Инсталлер предложит выполнить разбивку диска на разделы. Выбираем **Auto (UFS)** и продолжаем. Ни в коем случае не выбираем ZFS — это совсем не то, что требуется для нашей задачи!

5. Инсталлер отобразит список доступных дисков, среди которых будет **/dev/ada0** — это наш SSD накопитель, и предложит вариант разбивки на отдельные файловые системы. Ничего не меняем, соглашаемся и двигаемся дальше.

6. Инсталлер предложит выбрать тип таблицы разделов (BSD, GPT, MBR, и другие). Выбираем **GPT**. Это популярный и удобный способ, позволяет обращаться к томам диска по уникальным идентификаторам. *Эта аббревиатура не имеет ни малейшего отношения к нейросетям!*

7. После этого Инсталлер начнет установку: разметит диск, создаст таблицу разделов и распакует необходимые системные пакеты с ядром, утилитами и библиотеками. Весь процесс занимает от 30 до 60 секунд.

   Тут надо сделать замечание. Если на SSD находится ранее установленная другая операционная система, то Инсталлер попросить подтвердить факт того, что эта ОС будет уничтожена в процессе установки. Соглашаемся не глядя и нажимаем <**Commit**>.

8. После завершения установки, Инсталлер покажет черное терминальное окно и попросит задать пароль для пользователя **root**. Вводим несколько раз **1234**, какой-то серьезный и супер-сложный пароль тут не требуется. Мы готовим себе инструмент для работы, а не супер-секьюрный файерволл.

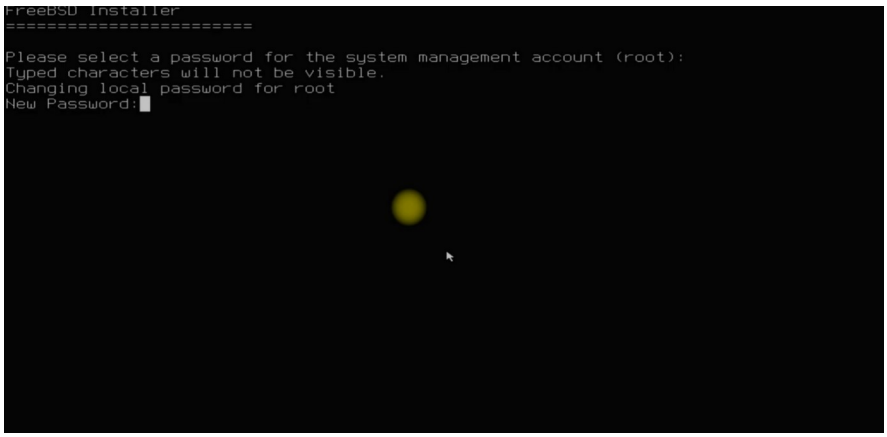

Рис.5. Установка пароля для пользователя root.

9. Инсталлер вернется к голубому экрану и попросит настроить сетевые интерфейсы. Производить настройку интерфейсов Ethernet не следует, мы выполним эту процедуру позже при настройке Ethernet Bridging-а. Но что следует сделать, так это настроить WiFi — это потребуется нам для удаленного управления устройством и установки дополнительного программного обеспечения.

   Необходимо выбрать интерфейс **wlan0** и ответить на дополнительные вопросы Инсталлера: указать имя (SSID) беспроводной сети, тип авторизации, пароль, указать на использование **IPv4** и включить **DHCP** для автоматического получения IP адреса и прочих сетевых настроек.

10. Инсталлер предложит настроить IPv6. Делать этого не следует, если конечно Вы не являетесь специалистом в этом вопросе. Выбираем **No** и двигаемся дальше.

11. Далее Инсталлер попросит указать имя домена к которому относится эта машина и имя DNS сервера. Оставляем всё как есть: **localdomain** и IP адрес DNS сервера полученного по DHCP. Если соединение с сетью на момент инсталляции отсутствует, то следует указать адрес DNS сервера вручную.

12. Следующим этапом Инсталлер попросит выбрать регион и страну для того, чтобы правильно установить временную зону. Выбираем **Asia**, потом **Russia**. Далее выбираем подходящую временную зону (в моём случае это **MSK+2**).

13. Далее система предложит установить текущую дату и время. Можно пропустить этот пункт (выбрать **Skip**), так как далее мы установим демона **ntpd** отвечающего за синхронизацию системного времени.

14. Следующим этапом Инсталлер предложит произвести ряд системных настроек. Необходимо отметить сервисы: **sshd** (удаленный доступ к системе), **ntpd** (синхронизация времени) и **powerd** (управление электропитанием).

15. Далее Инсталлер предложит усилить безопасность путем «завинчивания гаек» (System Hardening). Здесь ничего включать и выключать не требуется, в системе и так всё достаточно безопасно по умолчанию. К тому же, часть опций может добавить неожиданных проблем.

16. Следующим шагом Инсталлер предложит добавить нового пользователя. Вот это следует сделать обязательно! В появившемся черном экране укажите имя нового пользователя, например **sniffer**. Установите пароль, например **reffins**. Пользователя обязательно следует добавить в группы **wheel** и **network**, а также установить тип оболочки **tcsh**. См. скриншот на рис. 6.

    Добавление пользователя в группу **wheel** сделает этого пользователя администратором, а добавление в группу **network** позволить использовать сетевые утилиты без повышения уровня привилегий (без **sudo**).

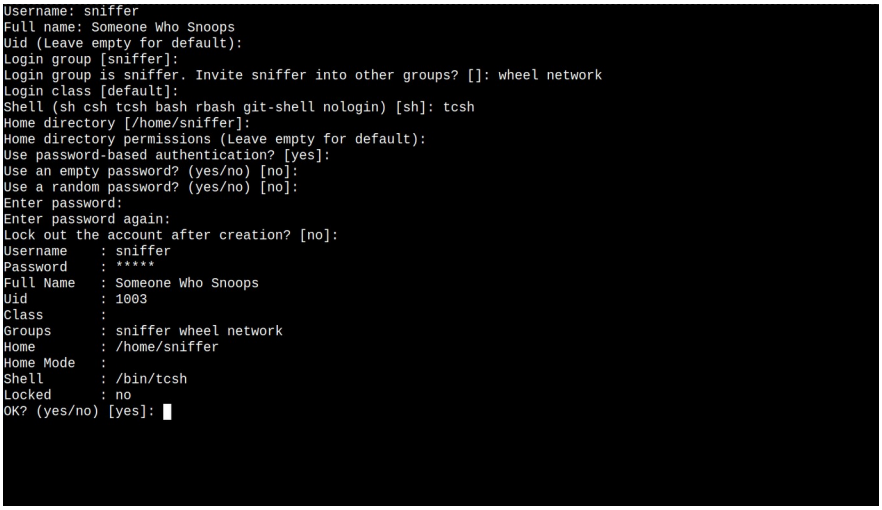

Рис.6. Добавление нового пользователя в систему с правами администратора.

17. После добавления нового пользователя Исталлятор вернется к голубому экрану и предложит выполнить еще ряд операций по настройке системы. На этом этапе можно закончить установку системы и перезагрузить машину. Для этого нужно вынуть USB Flash носитель из порта и выбрать меню **Exit**.

    Инсталлятор предложит запустить оболочку (shell) чтобы предоставить Вам еще одну возможность донастроить систему. Здесь нужно выбрать **No**, после чего Инсталлятор отобразит меню с опцией **Reboot**. Выбираем её и ждем пока система перезагрузится.

Пройдя через загрузчик и быстро пролетающий текст диагностических сообщений от ядра ОС, в конце концов пользователю предоставится приглашения для входа в систему:

```
Tue Jul  8 01:59:11 +05 2025

FreeBSD/amd64 (sniffer) (ttyv0)

login: sniffer
Password: reffins
```

Указываем пользователя **sniffer**, пароль **reffins** и попадаем в оболочку **tcsh** предварительно и очень кратко ознакомившись с «Message of the Day» (motd) следующего вида:

```
Welcome to FreeBSD!

Release Notes, Errata: https://www.FreeBSD.org/releases/
Security Advisories:   https://www.FreeBSD.org/security/
FreeBSD Handbook:      https://www.FreeBSD.org/handbook/
FreeBSD FAQ:           https://www.FreeBSD.org/faq/
Questions List:        https://www.FreeBSD.org/lists/questions/
FreeBSD Forums:        https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
FreeBSD directory layout:      man hier

To change this login announcement, see motd(5).

sniffer@sniffer:~ %
```

Последняя строка **sniffer@sniffer:~ %** это и есть приглашение от оболочки **tcsh**. Она означает, что Вы сейчас являетесь обычным пользователем (об этом говорит символ **%**) с именем **sniffer** на машине с именем **sniffer**. На этом этапе нам нужно продолжить доустановку различных полезных пакетов программ и утилит.

2.5. Установка полезных пакетов программ и утилит

Перед тем как начать установку пакетов, необходимо во-первых, получить привилегии администратора, а во-вторых выяснить настроено и поднято ли сетевое соединение через WiFi. Скачивание всех программ будем выполнять именно таким образом. Конечно же это можно сделать и через проводной Ethernet, но в данном случае будет удобнее, если проводные сетевые интерфейс **re0** и **re1** мы оставим нетронутыми для последующей настройки моста. Но это всё условности и Вы можете действовать иначе. :-)

Чтобы получить привилегии администратора, необходимо ввести команду **su** и указать пароль пользователя **root**. Напомню, что ранее при инсталляции пароль для root-а был задан как **1234**.

```
sniffer@sniffer:~ % su
Password: 1234

#
```

Если пароль введен верно, то строка приглашения ввода команд (промпт) изменит вид. Последний символ промпта **#** указывает на то, что Вы теперь администратор и должны действовать предельно аккуратно, чтобы случайно не удалить важных файлов и не испортить настройки системы!

Промпт в виде одного символа **#** предоставляет оболочка **sh**, она является дефолтной для пользователя root. Эта оболочка не очень удобна и я рекомендую сразу заменить её на оболочку **tcsh**.

Заменить оболочку можно следующей командой:

```
# chsh -s tcsh
chsh: user information updated
```

Утилита **chsh** сообщила, что оболочка заменена. Теперь выйдем из старой оболочки командой **exit** и еще раз выполним команду **su**:

```
# exit

sniffer@sniffer:~ % su

Password: 1234
root@sniffer:/home/sniffer #
```

Мы опять получили права администратора, но сейчас в промпте мы видим под каким пользователем (**root**) и на какой машине (**sniffer**) мы работаем, а также в каком каталоге дерева файловой системы (**/home/sniffer**) мы сейчас находимся. Это не единственное удобство предоставляемое оболочкой **tcsh**, нажимая стрелки вверх/вниз теперь можно перемещаться по истории ранее выполненных команд. При этом можно задавать часть имени команды и нажимать стрелки и **tcsh** найдет близкую по данной строке команду.

Проверить наличие соединения с WiFi можно командой **ifconfig**. Вводим команду:

```
root@sniffer:/home/sniffer # ifconfig wlan0
```

и получаем ответ вида:

```
wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=0
        ether b0:ac:82:7c:c2:5b
        inet 192.168.168.140 netmask 0xffffff00 broadcast 192.168.168.255
        groups: wlan
        ssid SOME_NETWORK channel 149 (5745 MHz 11a) bssid 50:ff:20:78:f8:23
        regdomain FCC country US authmode WPA1+WPA2/802.11i privacy ON
        deftxkey UNDEF AES-CCM 3:128-bit txpower 23 bmiss 7 mcastrate 6
        mgmtrate 6 scanvalid 60 wme roaming MANUAL
        parent interface: rtw880
        media: IEEE 802.11 Wireless Ethernet OFDM/36Mbps mode 11a
        status: associated
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

В выводе команды **ifconfig** интерес представляют выделенные строки:

- **status: associated** - означает, что соединение с WiFi точкой доступа установлено;
- **ssid SOME_NETWORK** — содержит имя точки доступа (здесь для примера: SOME_NETWORK);
- **inet 192.168.168.140** — показывает какой IP адрес выдан сетью (DHCP сервером) для данной машины.

Если у Вас получился подобный результат, значит соединение с сетью установлено, можно выполнить **ping** какого нибудь хоста в глобальной сети и проверить есть ли доступ в Интернет:

```
root@sniffer:/home/sniffer # ping www.google.com

PING www.google.com (173.194.220.147): 56 data bytes
64 bytes from 173.194.220.147: icmp_seq=0 ttl=107 time=106.200 ms
64 bytes from 173.194.220.147: icmp_seq=1 ttl=107 time=107.547 ms
64 bytes from 173.194.220.147: icmp_seq=2 ttl=107 time=106.863 ms
^C
--- www.google.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 106.200/106.870/107.547/0.550 ms
```

Выход из команды **ping**, как и из любой другой, в Unix-подобных системах осуществляется по нажатию **Ctrl-C.** Это инициирует посылку процессу сигнала SIGINT, что завершает его исполнение и управление передается обратно в оболочку **tcsh**.

Если с подключением к сети не задалось, то придется погрузиться в настройки WiFi. За работу WiFi в операционной системе FreeBSD отвечает демон **wpa_supplicant**, его файл конфигурации находится в **/etc/wpa_supplicant.conf**. Попробуйте проверить правильно ли указан пароль и имя точки доступа. Для этого откройте этот файл в редакторе **ee** или **vi**:

```
root@sniffer:/home/sniffer # ee /etc/wpa_supplicant.conf
```

После внесения исправлений в конфигурационный файл его следует сохранить. В редакторе **ee** это делается нажатием последовательности **ESC**, **Enter** и еще раз **Enter**. Сохраненный конфиг будет воспринят демоном **wpa_supplicant** только после рестарта сетевого (wlan0) интерфейса. Для этого вводим следующую команду:

```
root@sniffer:/home/sniffer # service netif restart wlan0
Stopping wpa_supplicant.
Waiting for PIDS: 46902.
Stopping Network: wlan0.
Destroyed wlan(4) interfaces: wlan0.
Created wlan(4) interfaces: wlan0.
Starting wpa_supplicant.
Starting Network: wlan0.
```

Через несколько секунд следует еще раз проверить наличие соединения командой **ifconfig wlan0**.

Будем предполагать, что в системе появился доступ в Интернет и мы можем перейти к установке пакетов. Установка, удаление и апгрейд собранных (заранее скомпилированных) пакетов программ в ОС FreeBSD осуществляется командой (утилитой) **pkg**. Эта утилита имеет несколько расширений:

- **pkg search** — осуществляет поиск пакета, по заданной строке символов, в централизованном репозитории расположенном на сайте проекта FreeBSD.
- **pkg install** — осуществляет установку пакета по его точному имени.
- **pkg remove** — удаляет ранее установленный пакет.
- **pkg info** — выдает подробную информацию об установленном пакете.

Посмотрим как это работает. Для примера, выполним поиск пакета по строке «WireShark»:

```
root@sniffer:/home/sniffer # pkg search WireShark
wireshark-4.4.7               Powerful network analyzer/capture tool
wireshark-nox11-4.4.7         Powerful network analyzer/capture tool (without GUI)
```

Команда **pkg search** выдает найденные пакеты и их краткое описание, по одному на строку. В данном примере мы видим, что в репозитории находятся два варианта пакета **wireshark** — с графическим интерфейсом и без такового (**nox11**).

Для задачи сетевого сниффера нам необходимо установить следующие пакеты программ и утилит:

- **sudo** — для получения привилегий администратора;
- **screen** — для управления виртуальными терминалами и одновременного запуска нескольких программ с удаленным управлением;
- **wireshark-nox11** — для захвата и анализа сетевого трафика;

- **trafshow** — для анализа текущей загрузки сети;
- **snort3** — для анализа и выявления потенциальных проблем безопасности и вторжения.
- **mpd5** — демон для создание туннельного соединение с сервером по L2TP или PPTP для организации удаленного канала управления снифферо.
- **gcc**, **cmake**, **gmake, git** — компилятор с языка Си и набор утилит для сборки других программ из исходных кодов.
- **cpu-x** — утилита для сбора и отображения информации об аппаратуре вычислительной системы.

Ну что же, давайте установим их все разом. Для этого введем следующую очень длинную команду:

```
root@sniffer:/home/sniffer # pkg install sudo screen wireshark-nox11 trafshow snort3 mpd5 gcc
cmake gmake git cpu-x

Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
Updating FreeBSD-kmods repository catalogue...
FreeBSD-kmods repository is up to date.
All repositories are up to date.
The following XXX package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        e2fsprogs-libuuid: 1.47.2 [FreeBSD]
        hwdata: 0.393,1 [FreeBSD]
        hwloc2: 2.11.2 [FreeBSD]
        ...
        screen: 4.9.1_5 [FreeBSD]
        snort3: 3.7.1.0,1 [FreeBSD]
        spdlog: 1.15.2 [FreeBSD]

Number of packages to be installed: 14

The process will require YYY MiB more space.
ZZZ MiB to be downloaded.

Proceed with this action? [y/N]: y
```

Утилита **pkg** проанализирует свою базу данных с пакетами, выявит все зависимости (другие библиотеки и приложения, которые требуются для работы тех, что мы просим установить) и отобразит длинный список пакетов для установки. В конце она сообщит объем требуемого дискового пространства и объем данных для скачивания из глобального репозитория. Жмем **y**, затем **Enter.** Утилита **pkg** выкачает пакеты, распакует их и установит в систему. В ходе процесса буду появляться сообщения вида:

```
[1/144] Fetching snort3-3.7.1.0,1.pkg: 100%    2 MiB   1.2MB/s    00:02
[2/144] Fetching luajit-devel-2.1.0.20250311.pkg: 100%  480 KiB 491.3kB/s    00:01
[3/144] Fetching hwloc2-2.11.2.pkg: 100%    3 MiB   1.5MB/s    00:02
[4/144] Fetching screen-4.9.1_5.pkg: 100%  568 KiB 581.8kB/s    00:01
...
[1/144] Installing screen-4.9.1_5...
[1/144] Extracting screen-4.9.1_5: 100%
[2/144] Installing snort3-3.7.1.0,1...
[2/144] Extracting snort3-3.7.1.0,1: 100%
[3/144] Installing e2fsprogs-libuuid-1.47.2...
[3/144] Extracting e2fsprogs-libuuid-1.47.2: 100%
[4/144] Installing hwloc2-2.11.2...
[4/144] Extracting hwloc2-2.11.2: 100%
...

root@sniffer:/home/sniffer #
```

Также могут появляться различные предупреждающие сообщения от конкретных пакетов, их можно смело игнорировать.

Если в процессе установки возникнут какие-то проблемы, т. е. сообщения об ошибках, и **pkg** остановится на пол-пути, то имеет смысл попробовать установить каждый пакет по отдельности, вводя команду **pkg install *имя_пакета***. Таким образом можно выяснить имя проблемного пакета и далее попытаться разобраться с ним индивидуально. Часто бывает, что имена пакетов меняются и нужно просто выяснить правильное название с помощью **pkg search**.

В конце мы опять увидим знакомый промпт. Это означает, что всё готово!

2.6. Подключение через SSH

Производить дальнейшую настройку систему через консоль (консолью традиционно называется клавиатура и монитор подключенные к системе напрямую) крайне неудобно, так как консоль не позволяет копировать куски текста или команды. Для того, чтобы упростить работу, следует подключиться к системе по сети, через WiFi, используя протокол удаленного доступа к оболочке — SSH (Secure Shell). Для этого необходимо иметь SSH клиент на рабочем компьютере. Если Вы являетесь пользователем Unix системы (Linux, *BSD), то проблем не возникнет — утилита **ssh** наверняка присутствует в Вашей системе. Для пользователей Windows необходимо установить SSH клиент который называется **putty.exe** (скачать его можно по ссылке), ну или подобный.

Перед тем как подключиться удаленно, через SSH, следует выяснить IP адрес машины сниффера в Вашей сети WiFi. Сделать это очень просто, выше мы уже рассматривали команду **ifconfig wlan0**, она выдает всю информацию о сетевом интрефейсе, в том числе назначенный IP адрес. В моем случае IP адрес машины - **192.168.168.140** поэтому для подключения к снифферу через SSH я использую команду с моего рабочего ПК:

```
% ssh sniffer@192.168.168.140
```

Для пользователей **putty.exe** все немного сложнее — потребуется создать новый сеанс, указать имя пользователя и IP адрес хоста, сохранить этот сеанс и только потом нажать **Connect**. Более подробно об использовании этой утилиты можно почитать на сайте проекта PuTTY.

В процессе подключения, при первом соединении, SSH клиент попросит подтвердить намерения установить соединение с указанным хостом и только после этого попросит ввести пароль. Вводим **reffins** и если всё прошло удачно, наблюдаем то же самое приглашение (motd) от системы, после которого следует всё тот же промпт:

```
sniffer@sniffer:~ %
```

Ради интереса спросим у системы кто мы есть и откуда, введем команду **who**:

```
sniffer@sniffer:~ % who
sniffer          pts/0        Jul 10 07:01 (192.168.171.1)
sniffer@sniffer:~ %
```

Получим привилегии администратора, для этого ведем:

```
sniffer@sniffer:~ % su
Password: 1234
root@sniffer:/home/sniffer #
```

Теперь клавиатуру и монитор можно смело отключить и спрятать в уголок (но не далеко), они нам, скорее всего, не понадобятся, так как работать дальше будем через SSH, но держать их стоит поблизости в качестве талисманов отгоняющих злых духов. ;)

2.7. Настройка `sudo’

Чтобы постоянно не находиться в оболочке с правами администратора (root-а), что может быть опасным из-за возможности случайно уничтожить систему неверно введенной командой, в Unix системах принято использовать утилиту **sudo** (или её аналог **doas**). Утилита **sudo** позволяет временно, для одной команды, которая передается ей в качестве параметров, поднять привилегии до администратора и сбросить их после завершения.

Мы только что установили утилиту **sudo** вместе с рядом других пакетов, но этого еще недостаточно для того, что бы пользователь **sniffer** мог ей воспользоваться. Нам требуется раскомментировать одну строку в конфигурационном файле **/usr/local/etc/sudo.conf**, но редактировать его мы будем через вспомогательную утилиту **visudo** (она является частью пакета **sudo**).

Сначала укажем системе, что далее для редактирования текстовых файлов следует использовать редактор **ee**. Делается это установкой переменной окружения **EDITOR**:

```
root@sniffer:/home/sniffer # setenv EDITOR /usr/bin/ee
```

Теперь вызываем **visudo** для редактирования конфигурационного файла:

```
root@sniffer:/home/sniffer # visudo
```

Утилита **visudo** запустит редактор **ee** и загрузит в него текущее содержимое конфигурационного файла. *Здесь надо быть предельно осторожным чтобы случайно не изменить ничего лишнего!* Находим в тексте строку вида:

```
## Uncomment to allow members of group wheel to execute any command
# %wheel ALL=(ALL:ALL) ALL
```

и перед символами **%wheel** убираем символ **#**. Это означает, что данная строка начнет действовать. Её действие сводится к тому, что утилита **sudo** позволит любому пользователю входящему в группу **wheel** исполнять любую команду от имени root-а. Напомню, что когда мы создавали в системе пользователя **sniffer**, мы добавляли его в группы **wheel** и **network**. В результате редактирования строка должна выглядеть вот так:

```
%wheel ALL=(ALL:ALL) ALL
```

Нажимаем **ESC**, **Ener** и еще раз **Enter** чтобы сохранить изменения в файл и выйти из **ee**. С этого момента **sudo** позволит пользователю **sniffer** исполнять команды от имени пользователя **root**, то есть с правами администратора.

Чтобы выяснить в какие группы входит пользователь **sniffer**, можно воспользоваться командой **id** следующим образом:

```
root@sniffer:/home/sniffer # id sniffer
uid=1002(sniffer) gid=1002(sniffer) groups=1002(sniffer),0(wheel),69(network)
```

Команда **id** показывает все системные идентификаторы указанного пользователя. Здесь мы видим, что пользователь **sniffer** входит аж в три группы: **sniffer**, **wheel** и **network**. Обратите внимание, что группа **wheel** имеет идентификатор **0** — это группа администраторов.

Если вдруг Вы забыли добавить пользователя **sniffer** в нужную группу, сделать это можно в любой момент командой **pw usermod**:

```
root@sniffer:/home/sniffer # pw usermod sniffer -G wheel,network
```

Эта команда устанавливает (замещает) список групп для пользователя **sniffer**.

Проверим как работает **sudo**. Для этого сначала выйдем из администраторской оболочки введя команду **exit**, то есть вернемся в оболочку без привилегий:

```
root@sniffer:/home/sniffer # exit
exit
sniffer@sniffer:~ %
```

Далее, исполним команду **id** от имени администратора:

```
sniffer@sniffer:~ % sudo id

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

For security reasons, the password you type will not be visible.
Password: reffins
uid=0(root) gid=0(wheel) groups=0(wheel),5(operator)
```

Сначала **sudo** предупредит о том, что Вы пытаетесь получить привилегии администратора, что несет за собой определенную ответственность и Вам следует действовать осторожно. Данное предупреждение появится только один раз при первом использовании. Далее **sudo** запросит у Вас пароль от пользователя **sniffer**. Если введенный пароль верен, то команда **id** исполняется с правами пользователя **root**. Это видно по идентификаторам **uid=0(root) gid=0(wheel)** отображаемых этой командой.

Если вдруг **sudo** выдаст сообщение вида:

```
sniffer is not in the sudoers file.
This incident has been reported to the administrator.
```

Это означает, что либо пользователь **sniffer** не входит в группу **wheel**, либо в конфигурационном файле есть ошибка. Стоит еще раз запустить **visudo** и проверить наличие строки **%wheel**.

Далее мы будем работать в оболочке от пользователя **sniffer** без привилегий, промпт должен выглядеть вот так: **sniffer@sniffer:~ %**. А когда понадобится выполнить команду требующую привилегий администратора, то будем использовать **sudo**.

3. Исследуем аппаратную часть

3.1. Получаем сведения об аппаратуре

Перед тем как перейти к сетевым настройкам, следует немного изучить имеющееся у нас «железо». Как уже отмечалось выше, одноплатный мини-ПК Horizon N5 оснащен микропроцессором Intel Celeron N5105. Посмотрим, что наша система знает про него. Для этого воспользуемся чудесной утилитой **cpu-x**. У этой утилиты есть два режима работы: интерактивный и командный.

Если запустить **cpu-x** без параметров, то она перейдет в интерактивный режим с многоэкранным текстовым меню. Перемещаясь по меню стрелками курсора можно в реальном времени наблюдать за изменениями тактовой частоты центрального процессора, скоростей работы шины, кэшей и оперативной памяти.

Чтобы получить всю информацию сразу, можно указать ключ **-D**. В обоих случаях запускать утилиту **cpu-x** следует с привилегиями администратора, например:

```
sniffer@sniffer:~ % sudo cpu-x
Password: reffins
```

После чего отобразится окно с текстовым меню как на рис. 7.

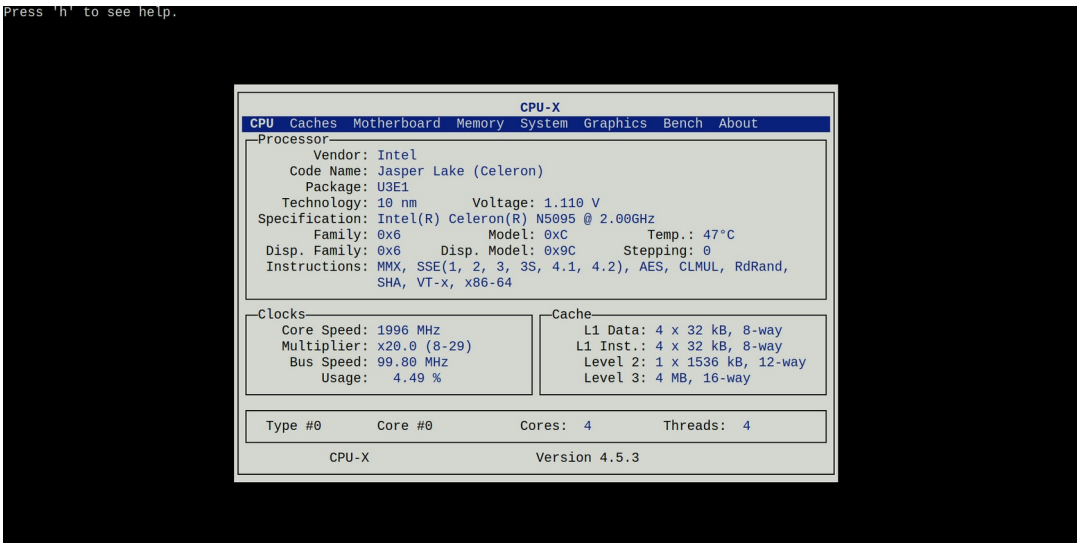

Рис.7. Информационное окно утилиты cpu-x на мини-ПК Rombic Horizon N5.

Либо в командном режиме, чтобы получить полный листинг с описанием аппаратуры. Введем команду:

```
sniffer@sniffer:~ % sudo cpu-x -D


 >>>>>>>>>> CPU <<<<<<<<<<

      ***** Processor *****
        Vendor: Intel
     Code Name: Jasper Lake (Celeron)
       Package: U3E1
    Technology: 10 nm
       Voltage: 1.110 V
 Specification: Intel(R) Celeron(R) N5095 @ 2.00GHz
        Family: 0x6
```

```
  Disp. Family: 0x6
         Model: 0xC
   Disp. Model: 0x9C
         Temp.: 48°C
      Stepping: 0
  Instructions: MMX, SSE(1, 2, 3, 3S, 4.1, 4.2), AES, CLMUL, RdRand, SHA, VT-x, x86-64

      ***** Clocks *****
    Core Speed: 1996 MHz
    Multiplier: x20.0 (8-29)
     Bus Speed: 99.80 MHz
         Usage:   0.39 %

      ***** Cache *****
      L1 Data: 4 x 32 kB, 8-way
    L1 Inst.: 4 x 32 kB, 8-way
      Level 2: 1 x 1536 kB, 12-way
      Level 3: 4 MB, 16-way

      ***** * *****
         Cores: 4
       Threads: 4


>>>>>>>>>> Caches <<<<<<<<<<

      ***** L1 Cache *****
          Size: 4 x 32 kB, 8-way associative, 64-bytes line size
         Speed: 83410.70 MB/s

      ***** L2 Cache *****
          Size: 1 x 1536 kB, 12-way associative, 64-bytes line size
         Speed: 35808.30 MB/s

      ***** L3 Cache *****
          Size: 4 MB, 16-way associative, 64-bytes line size
         Speed: 24022.30 MB/s


>>>>>>>>>> Motherboard <<<<<<<<<<

      ***** Motherboard *****
  Manufacturer: Rombica
         Model: Default string
      Revision: Default string

      ***** BIOS *****
         Brand: American Megatrends International, LLC.
       Version: T9PRO001
          Date: 10/23/2024
      ROM Size: 64 kB / 5 MB

      ***** Chipset *****
        Vendor: Intel Corporation
         Model: Jasper Lake eSPI Controller


>>>>>>>>>> Memory <<<<<<<<<<

      ***** Bank 0 *****
     Reference: Samsung                  , 6 GB @ 3733 MT/s (Row Of Chips LPDDR4)

      ***** Bank 1 *****
     Reference: Samsung                  , 6 GB @ 3733 MT/s (Row Of Chips LPDDR4)


>>>>>>>>>> System <<<<<<<<<<

      ***** Operating System *****
        Kernel: 14.3-RELEASE
  Distribution: FreeBSD
      Hostname: sniffer
        Uptime: 0 days, 1 hours, 54 minutes, 50 seconds
      Compiler: FreeBSD clang version 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg

      ***** Memory *****
```

```
          Used: 0.90 GiB / 11.46 GiB
       Buffers: 0.00 GiB / 11.46 GiB
        Cached: 0.00 GiB / 11.46 GiB
          Free: 10.55 GiB / 11.46 GiB
          Swap: 0.00 GiB / 4.00 GiB


>>>>>>>>>> Graphics <<<<<<<<<<

      ***** Card 0 *****
        Vendor: Intel
        Driver:
   UMD Version:
         Model: JasperLake [UHD Graphics]
  Compute Unit:
      DeviceID: 0x4E55:0x01
 VBIOS Version:
     Interface:
   Temperature:
         Usage:
  Core Voltage:
     Power Avg:
     GPU clock:
  Memory clock:
   Memory Used:
 Resizable BAR:
     Vulkan RT:
OpenGL Version:
Vulkan Version:
OpenCL Version:
```

3.2. Проводим тест Coremark для вычислительного ядра

Тест **Coremark** позволяет измерить пиковую производительность системы в некоторых относительных единицах, которые выражаются в числе итераций основного цикла выполняемых за секунду времени. Зная этот параметр для испытуемой системы можно сопоставить её с другими известными вычислительными системами и комплексами и приблизительно оценить нагрузку, с которой может справиться эта система. Тест Coremark пришел на смену тесту Dhrystone и считается более релевантным для современных вычислительных систем. Тест Coremark представляет собой программу, написанную на языке Си, и распространяется он в исходных кодах. Эту программу перед запуском требуется собрать (скомпилировать), что можно выполнить с различными параметрами оптимизации, а следовательно тест может показывать очень сильно отличающиеся результаты. Поэтому, при сравнении результатов всегда необходимо обращать внимание на опции компиляции.

Получить исходные коды теста можно из репозитория с Github-а следующей командой:

```
sniffer@sniffer:~ % git clone https://github.com/eembc/coremark.git
Cloning into 'coremark'...
remote: Enumerating objects: 412, done.
remote: Counting objects: 100% (175/175), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 412 (delta 136), reused 115 (delta 105), pack-reused 237 (from 2)
Receiving objects: 100% (412/412), 542.69 KiB | 1.51 MiB/s, done.
Resolving deltas: 100% (234/234), done.
```

Привилегии администратора при этом не требуются. Сборка и запуск теста выполняется сразу одной командой:

```
sniffer@sniffer:~ % cd coremark && gmake
```

При этом будет выполнены два прогона теста с несколько разными параметрами. Результаты тестов будут помещены в файлы **./run1.log** и **./run2.log**. Нас интересует первый прогон (performance). Чтобы отобразить результат воспользуемся командой **cat**:

```
sniffer@sniffer:~/coremark % cat run1.log
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 16601
Total time (secs): 16.601000
Iterations/Sec   : 18071.200530
Iterations       : 300000
Compiler version : FreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg-19.1.7-0-gcd708029e0b2)
Compiler flags   : -O2 -DPERFORMANCE_RUN=1  -lrt
Memory location  : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xcc42
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 18071.200530 / FreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg-19.1.7-0-gcd708029e0b2) -O2 -DPERFORMANCE_RUN=1  -lrt / Heap
```

Интересующая нас цифра выделена жирным и составляет **18071** единиц. Часто этот показатель переводят в нормированные единицы на 1 МГц. В нашем случает это число следует поделить на тактовую частоту 1999. Получим **9.0 единиц Coremark/MHZ**.

Важно отметить, что по умолчанию программа тестирует только одно вычислительное ядро. Чтобы проверить суммарную производительность всех ядер, необходимо распараллелить тест на число ядер микропроцессора. Как сообщает утилита **cpu-x**, микропроцессор Intel Celeron N5095 имеет 4 параллельных нити исполнения программ. Выполним тест на всех четырех нитях:

```
sniffer@sniffer:~/coremark % gmake clean
rm  -f  ./coremark.exe  ./core_list_join.o  ./core_main.o  ./core_matrix.o  ./core_state.o
./core_util.o ./*.log *.info ./index.html

sniffer@sniffer:~/coremark % gmake XCFLAGS="-DMULTITHREAD=4 -DUSE_FORK"

sniffer@sniffer:~/coremark % cat run1.log
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 16682
Total time (secs): 16.682000
Iterations/Sec   : 71933.820885
Iterations       : 1200000
Compiler version : FreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg-19.1.7-0-gcd708029e0b2)
Compiler flags   : -O2 -DMULTITHREAD=4 -DUSE_FORK -DPERFORMANCE_RUN=1  -lrt
Parallel Fork : 4
Memory location  : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[1]crclist       : 0xe714
[2]crclist       : 0xe714
[3]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[1]crcmatrix     : 0x1fd7
[2]crcmatrix     : 0x1fd7
[3]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[1]crcstate      : 0x8e3a
[2]crcstate      : 0x8e3a
[3]crcstate      : 0x8e3a
```

```
[0]crcfinal      : 0xcc42
[1]crcfinal      : 0xcc42
[2]crcfinal      : 0xcc42
[3]crcfinal      : 0xcc42
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 71933.820885 / FreeBSD Clang 19.1.7 (https://github.com/llvm/llvm-project.git
llvmorg-19.1.7-0-gcd708029e0b2) -O2 -DMULTITHREAD=4 -DUSE_FORK -DPERFORMANCE_RUN=1  -lrt /
Heap / 4:Fork
```

В результате получаем **71933 Coremark** на все четыре ядра.

Чтобы понять, много это или мало, можно сравнить с аналогичными измерениями выполненными для известных систем. Готовые измерения можно просмотреть по ссылке на сайте EEMBC. Для сравнения, **RaspberryPI 5** с тактовой частотой **2.5 МГц** выдает **72059** единиц, а мой ноутбук Lenovo с процессором **AMD Ryzen5** и частотой **3.3 ГГц** выдает **303116** единиц Coremark. Очевидно, что имеющийся в нашем распоряжении мини-ПК не уступает популярному одноплатному ПК RaspberryPI 5, но почти на порядок отстает от серьезных десктопных систем.

3.3. Проводим тест STREAM для оперативной памяти

Давно известно, что узким местом в современных вычислительных системах является именно память, её недостаточная скорость работы может радикально снизить производительность всей системы. Тест STREAM состоит из четырех операций: измерение производительности операции копирования (copy), масштабирования (scale), сложения содержимого ячеек (add) и комплексных операций: масштабирование+сложения+пересылки (triad). Как и тест Coremark, тест STREAM является устоявшимся промышленным стандартом, более подробно прочитать о нём можно по ссылке: https://documentation.sigma2.no/jobs/arm-perf/stream.html

Тест STEAM также распространяется в исходных кодах, поэтому чтобы выполнить замеры производительности памяти на нашем устройстве, аналогичным образом клонируем репозиторий:

```
sniffer@sniffer:~ % git clone https://github.com/jeffhammond/STREAM.git

Cloning into 'STREAM'...
remote: Enumerating objects: 89, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 89 (delta 31), reused 24 (delta 24), pack-reused 51 (from 2)
Receiving objects: 100% (89/89), 32.88 KiB | 320.00 KiB/s, done.
Resolving deltas: 100% (50/50), done.
```

Перейдем в подкаталог с исходным кодом и выполним сборку:

```
sniffer@sniffer:~ % cd STREAM && gmake
```

В результате сборки у нас образуется исполняемый файл с именем **stream_c.exe.** Запустим его:

```
sniffer@sniffer:~/STREAM % ./stream_c.exe

-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
```

```
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 4
Number of Threads counted = 4
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 7411 microseconds.
   (= 7411 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           14527.0     0.011080     0.011014     0.011218
Scale:          13139.5     0.012274     0.012177     0.012343
Add:            12786.4     0.018830     0.018770     0.018882
Triad:          12819.8     0.018799     0.018721     0.018890
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

В результате мы видим, что операции копирования выполняются со скоростью около **14 ГБ/сек**. Для сравнения, на моём ноутбуке Lenovo этот показатель достигает **18 ГБ/сек**, но надо отметить что в Horizon N5 используется высокоскоростная память LPDDR4, такая же как и в моём ноуте, поэтому результаты очень близки. На серьезных серверных платформах тест STREAM может показывать 120 ГБ/сек (и даже более) для операций копирования.

Еще один интересный момент и побочный результат измерения производительности с помощью STREAM это коэффициент ошибок, который показывает надежность всей подсистемы оперативной памяти. В приведенном примере тест показывает, что коэффициент ошибок составляет менее **1.000000e-13**, то есть их вообще не зафиксировано. Хороший признак!

С аппаратной частью разобрались и сделали соответствующие выводы. Вычислительных ресурсов системы вполне достаточно чтобы проводить несложные замеры трафика и даже анализировать его «на лету».

4. Настойка канала для удаленного доступа к устройству (mpd5)

Следующим этапом в настройке сниффера считаю необходимым рассмотрение вопроса удаленного доступа к машине и тут есть различные варианты. Удаленный доступ к снифферу мы будем осуществлять по протоколу SSH, так как это самый традиционный и повсеместно используемый способ удаленного управления Unix машиной, хотя я уверен что пользователи Windows предпочли бы использовать RDP. RDP, кстати, тоже можно настроить, но сегодня не их день. :)

В Unix системах подключение по SSH осуществляется одноименной командой **ssh** входящей в пакет OpenSSH. Этот пакет является частью многих операционных систем и уже установлен в процессе инсталляции. Команда **ssh** принимает на вход обязательный параметр вида username@hostname или username@x.x.x.x где x.x.x.x — IP адрес машины. С username всё понятно, это имя пользователя в системе, в нашем случает это пользователь **sniffer**. А вот с IP адресом (и тем более с именем хоста, которое можно преобразовать в IP адрес) есть ряд проблем. Как нам узнать какой IP адрес у машины сниффера когда она установлена, например, у абонента в квартире ? Более того, если даже мы узнаем этот IP, то толку от этого знания нет никакого, так как в домашней сети абонента всегда используются «фэйковые», т.е. публично не маршрутизируемые IP адреса, а сам абонент находится за одним и более транслятором адресов (NAT — Network Adress Translation). Всё это НЕ позволяет установить соединение с хостом сниффера расположенного внутри домашней сети абонента находясь за её пределами. Но что же делать ?

Решений несколько. Можно воспользоваться средствами самого **ssh** для организации обратного туннеля (SSH Reverse Tunneling) с помощью опции **-R** к известному и публично доступному Unix хосту, к которому у Вас есть доступ. Это решение не простое в том плане, что требует навыков в использовании Unix систем и понимания происходящего процесса, поэтому рассматривать здесь его не будем. Желающие могут ознакомиться с инструкцией по настройке обратного туннеля в сети Интернет.

В место этого рассмотрим более традиционный способ — организация туннеля к известному публичному хосту по протоколам PPTP (или L2TP) известных широкой массе как Virtual Private Network (VPN). Эта технология позволяет объединять разбросанные по глобальной Сети хосты компании в единую сеть. Достоинство VPN технологии состоит в том, что Вам не потребуется организовывать свой PPTP/L2TP сервер, достаточно обратиться к ближайшему провайдеру услуг Интернет с просьбой организовать для Ваших служебных нужд аккаунт с фиксированным публичным IP адресом. Такая услуга будет не бесплатной, но зато снимает массу проблем. Разумеется, Вы так же можете арендовать у Интернет провайдера виртуальный сервер с ОС Linux и организовать на нём свой служебный VPN сервер, но это тема отдельного разговора.

Короче, будем полагать, что проблема получения аккаунта к серверу VPN каким-то чудесным образом разрешена и у Вас имеется четыре строки текста: IP адрес PPTP сервера, пусть это будет **1.2.3.4**, имя пользователя: **pptp_username** и пароль: **pptp_password**. Также мы знаем, что после поднятия соединения с сервером на стороне клиента всегда будет установлен один и тот же публичный IP адрес **4.3.2.1**.

Наша задача на данном этапе состоит в том, чтобы настроить PPTP или L2TP клиента в на машине сниффера. Надо заметить, что организацией туннелей по протоколам PPTP и L2TP в операционной системе FreeBSD занимается демон (фоновая утилита) **mpd5,** установку которой уже выполнили ранее. На самом деле всю работу по инкапсуляции трафика выполняет подсистема **netgraph** (введите команду **man 4 netgraph** для более

детального ознакомления), а **mpd5** это всего лишь надводная часть этого айсберга выполняющая авторизацию и конфигурирование сетевого интерфейса.

Все настройки демона **mpd5** находятся в каталоге **/usr/local/etc/mpd5/**, обычно там располагаются всего три файла: **mpd.conf**, **mpd.script** и **mpd.secret**. Нам потребуется внести изменения в файл **mpd.conf** — добавить секцию с описанием соединения, и создать два новых файла-скрипта. Один скрипт (назовем его **ifup.sh**) для добавления специфических маршрутов в таблицу маршрутизации при установке соединения, а другой (**ifdown.sh**) для удаления их при разъединении. А еще нам потребуется внести изменения в системный файл конфигурации операционной системы **/etc/rc.conf** чтобы выполнялся авто-запуск **mpd5** при загрузке ОС.

Используя редактор **ee** от имени root-а открываем файл **/usr/local/etc/mpd5/mpd.conf** командой**:**

```
sniffer@sniffer:~ % sudo ee /usr/local/etc/mpd5/mpd.conf
```

Копируем в буфер обмена и вставляем в самое начало файла следующий текст:

```
startup:
        log +ALL +EVENTS -FRAME -ECHO

default:
        load PPTP

L2TP:
        create bundle static B1

        set iface up-script /usr/local/etc/mpd5/ifup.sh
        set iface down-script /usr/local/etc/mpd5/ifdown.sh

        set iface enable tcpmssfix
        set ipcp yes vjcomp

        set ccp yes mppc
        set mppc yes e128
        set mppc yes stateless

        create link static L1 l2tp
        set link action bundle B1
        set link max-redial 0
        set link mtu 1460
        set link keep-alive 20 75
        set link accept chap-msv2

        set l2tp peer 1.2.3.4
        set auth authname "l2tp_username"
        set auth password "l2tp_password"

        open

PPTP:

        create bundle static B1
        set bundle enable compression
        set ccp yes mppc
        set mppc no e40
        set mppc yes e128
        set mppc yes stateless

        set iface up-script /usr/local/etc/mpd5/ifup.sh
        set iface down-script /usr/local/etc/mpd5/ifdown.sh

        create link static L1 pptp
        set link action bundle B1
        set link max-redial 0
        set link mtu 1460
        set link keep-alive 20 75
        set pptp disable windowing
```

```
set pptp peer 1.2.3.4
set auth authname "pptp_username"
set auth password "pptp_password"

open
```

Перед сохранением необходимо подменить IP адрес сервера, имя пользователя и пароль, а также выбрать какой из двух протоколов будет использоваться. Находим следующие строки:

```
set l2tp peer 1.2.3.4
set auth authname "l2tp_username"
set auth password "l2tp_password"

set pptp peer 1.2.3.4
set auth authname "pptp_username"
set auth password "pptp_password"
```

и заменяет значения параметров на действительные, полученные от Интернет-провайдера.

Для выбора протокола в самом начале ищем строки:

```
default:
        load PPTP
```

и заменяем **load PPTP** на **load L2PT** если требуется использовать протокол L2PT. Иначе оставляем как есть.

Выходим из редактора **ee**, как обычно, последовательным нажатием: **ESC**, **Enter**, **Enter**.

Следующим шагом создаем файлы скриптов для поднятия сетевого интерфейса. Открываем в редакторе **ee** сначала файл **ifup.sh**:

```
sniffer@sniffer:~ % sudo ee /usr/local/etc/mpd5/ifup.sh
```

Копируем и вставляем в него следующий текст:

```
#!/bin/sh
echo $* > /tmp/.mpd5.$1
wanip=`/sbin/route -n get default | sed -rn 's/gateway: (.*)/\1/p'`
echo $wanip > /tmp/.defaultgateway
localip=$3
remoteip=$4
serverip=$8
echo localip=$3 remoteip=$4 serverip=$8 >> /tmp/.mpd5.$1
/sbin/route delete $serverip
/sbin/route add $serverip $wanip
/sbin/route delete default
/sbin/route add default $remoteip
```

Сохраняем файл (ESC, Enter, Enter) и следом открываем другой файл **ifdown.sh**:

```
sniffer@sniffer:~ % sudo ee /usr/local/etc/mpd5/ifdown.sh
```

Копируем и вставляем в него следующий текст:

```
#!/bin/sh
wanip=`cat /tmp/.defaultgateway`
/sbin/route delete $4
/sbin/route delete default
```

```
/sbin/route add default $wanip
rm /tmp/.defaultgateway
```

И тоже сохраняем.

Теперь нам нужно добавить **mpd5** в авто-запуск, чтобы при загрузки операционной системы эта утилита запускалась и немедленно устанавливала соединение с VPN сервером, таким образом организуя нам канал доступа на машину.

В ОС FreeBSD процесс авто-запуска программ, да и многие другие аспекты загрузки системы, контролируются набором скриптов которые принято называть **rc** — по имени главного скрипта. Настройки авто-запуска располагаются в файле **/etc/rc.conf**. Этот файл можно редактировать обычным текстовым редактором (**ee**), но так делать не принято по вполне понятной причине — незаметная ошибка в этом файле чревата выводом системы из оборота. Для того, чтобы вносить изменения в конфигурационные файл безопасным способом служит отдельная утилита **sysrc**.

Воспользуемся утилитой **sysrc** для добавления **mpd5** в **/etc/rc.conf**, выполним следующие команды:

```
sniffer@sniffer:~ % sudo sysrc mpd_enable="YES"
Password:
mpd_enable:  -> YES

sniffer@sniffer:~ % sudo sysrc mpd_flags="-b"
mpd_flags:  -> -b
```

Последняя команда добавляет в **/etc/rc.conf** флаг **-b** который будет передан утилите **mpd5** при старте и сигнализирует ей о необходимости работать в фоновом режиме (то есть демоном).

Теперь можем запустить **mpd5** и посмотреть что у нас получилось. Запуск системных демонов, то есть тех, что управляемы через **rc**, в ОС FreeBSD осуществляется утилитой **service**. Выполним запуск **mpd5** следующей командой:

```
sniffer@sniffer:~ % sudo service mpd5 start
Starting mpd5.
```

Если все прошло гладко, т.е. в конфигах нет опечаток, а все настройки (IP адрес сервера, имя пользователя и пароль) корректны, то в системе должен появится еще один сетевой интерфейс с именем **ng0**. Просмотреть его параметры можно командой **ifconfig ng0** (для просмотра права root-а не требуются):

```
sniffer@sniffer:~ % ifconfig ng0
ng0: flags=10088d1<UP,POINTOPOINT,RUNNING,NOARP,SIMPLEX,MULTICAST,LOWER_UP> metric 0 mtu
1456
        options=0
        inet 192.168.176.135 --> 192.168.176.1 netmask 0xffffffff
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Здесь мы видим, что сервер выдал мне IP адрес 192.168.176.135. В Вашем случае здесь должен высвечиваться фиксированный публичный IP адрес который выдан Вам провайдером, т. е. **4.3.2.1**.

Если что-то пошло не так, и вместо IP адреса на интерфейс ничего нет, то есть как-то вот так:

```
sniffer@sniffer:~ % ifconfig ng0
ifconfig: interface ng0 does not exist
```

Это означает, что **mpd5** не запущен или даже не установлен. Или вот так:

```
sniffer@sniffer:~ % ifconfig ng0
ng0: flags=8890<POINTOPOINT,NOARP,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=0
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Это означает, что **mpd5** запустился, но установить соединение по PPTP или L2TP у него не получилось. Чтобы выяснить причину необходимо заглянуть в лог файл **/var/log/daemon.log**, в него утилита **mpd5** выдает огромное количество диагностической информации. С помощью утилиты **grep** выберем информацию из этого файла касаемую установки соединения:

Например:

```
sniffer@sniffer:~ % grep mpd /var/log/daemon.log

...
Jul 10 09:04:48 sniffer mpd[2986]: [L1] CHAP: Using authname "Sniffer"
Jul 10 09:04:48 sniffer mpd[2986]: [L1] CHAP: sending RESPONSE #1 len: 61
Jul 10 09:04:48 sniffer mpd[2986]: [L1] CHAP: rec'd FAILURE #1 len: 31
Jul 10 09:04:48 sniffer mpd[2986]: [L1]   MESG: E=691 R=0 M=Login incorrect
Jul 10 09:04:48 sniffer mpd[2986]: [L1] LCP: authorization failed
Jul 10 09:04:48 sniffer mpd[2986]: [L1] LCP: parameter negotiation failed
Jul 10 09:04:48 sniffer mpd[2986]: [L1] LCP: state change Opened --> Stopping
Jul 10 09:04:48 sniffer mpd[2986]: [L1] LCP: phase shift AUTHENTICATE --> TERMINATE
Jul 10 09:04:48 sniffer mpd[2986]: [L1] AUTH: Cleanup
...
```

Из этого лога видно что была произведена попытка использовать имя пользователя **Sniffer**, что привело к сообщению «**Loging incorrect»** из-за опечатки в первом символе. Необходимо открыть файл **/usr/local/etc/mpd5/mpd.conf** в редакторе **ee** и исправить опечатку, после чего выполнить команду перезапуска **mpd5**:

```
sniffer@sniffer:~ % sudo service mpd5 restart
Stopping mpd5.
Waiting for PIDS: 2986.
Starting mpd5.
```

и тут же проверить:

```
sniffer@sniffer:~ % ifconfig ng0
ng0: flags=10088d1<UP,POINTOPOINT,RUNNING,NOARP,SIMPLEX,MULTICAST,LOWER_UP> metric 0 mtu
1456
        options=0
        inet 192.168.176.135 --> 192.168.176.1 netmask 0xffffffff
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Если всё ок, IP адрес на интерфейсе **ng0** присутствует, то можно подключаться к снифферу на этот IP адрес следующей командой со своего рабочего ПК:

```
$ ssh sniffer@4.3.2.1
```

где **4.3.2.1** — IP адрес на туннельном интерфейсе, он же назначенный Вашим провайдером. В моем случае это адрес **192.168.176.135**, он является локальным для моей сети и использован только для демонстрации.

С этого момента у Вас появляется возможность удаленного подключения к снифферу через глобальную сеть Интернет, а пароль пользователя **sniffer** следует поменять на что-то более безопасное или сделать авторизацию по несимметричным ключам.

5. Настройка сетевого «моста» (Ethernet Bridging)

В ОС FreeBSD поднять сетевой мост можно между любым числом сетевых интерфейсов, более того, они не обязательно должны быть типа Ethernet. Но нас в данном контексте интересует мост из двух встроенных Etherent портов, которые представляются под определенными именами. Чтобы выяснить какие сетевые интерфейсы имеются в системе воспользуемся системной утилитой **ifconfig**, введем команду:

```
sniffer@sniffer:~ % ifconfig -l
re0 re1 lo0 wlan0 ng0
```

В результате получим простой список имен интерфейсов разделенных пробелом. В данном случае интерфейсы **re0** и **re1** это и есть Ethernet порты. Более подробно информацию о них можно получить также с помощью **ifconfig**:

```
sniffer@sniffer:~ % ifconfig re0
re0: flags=8902<BROADCAST,PROMISC,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8209b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,WOL_MAGIC,LINKSTATE>
        ether 68:1d:ef:47:de:0a
        media: Ethernet autoselect (none)
        status: no carrier
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>

sniffer@sniffer:~ % ifconfig re1
re1: flags=8902<BROADCAST,PROMISC,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8209b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,WOL_MAGIC,LINKSTATE>
        ether 68:1d:ef:47:de:0b
        media: Ethernet autoselect (none)
        status: no carrier
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Утилита **ifconfig** отображает большой объем различной информации о сетевом интерфейсе, среди которой нас может заинтересовать MAC адрес (**ether 68:1d:ef:47:de:0** и **ether 68:1d:ef:47:de:0b**), а также наличие соединения с коммутатором или другим устройством (**status: no carrier** говорит об отсутствии соединение, дословно: «нет несущей частоты»).

5.1. Загрузка драйверов if_bridge и bridgestp

Информацию о том как настроить Brigding (сетевой мост) можно легко получить из встроенного системного руководства. Давайте заглянем в него введя следующую команду:

```
sniffer@sniffer:~ % man bridge

IF_BRIDGE(4)           FreeBSD Kernel Interfaces Manual           IF_BRIDGE(4)

NAME
     if_bridge – network bridge device

SYNOPSIS
     To compile this driver into the kernel, place the following line in your
     kernel configuration file:

           device if_bridge

     Alternatively, to load the driver as a module at boot time, place the
     following lines in loader.conf(5):

           if_bridge_load="YES"
           bridgestp_load="YES"

DESCRIPTION
```

```
The if_bridge driver creates a logical link between two or more IEEE 802
networks that use the same (or “similar enough”) framing format.  For
example, it is possible to bridge Ethernet and 802.11 networks together,
but it is not possible to bridge Ethernet and Token Ring together.
```

На первой же странице руководства нам предлагается добавить два системных модуля (драйвера) в конфигурационный файл загрузчика **/boot/loader.conf**. Сделать это можно с помощью утилиты **sysrc** указав после ключа **-f** имя конфигурационного файла:

```
sniffer@sniffer:~ % sudo sysrc -f /boot/loader.conf if_bridge_load="YES"
Password: reffins
if_bridge_load:  -> YES

sniffer@sniffer:~ % sudo sysrc -f /boot/loader.conf bridgestp_load="YES"
bridgestp_load:  -> YES
```

Первый драйвер, **if_bridge**, собственно реализует требуемый нам функционал сетевого моста. Второй драйвер, **bridgestp**, реализует протокол Rapid Spanning Tree Protocol (RSTP, 802.1w) для мостового соединения позволяющий избежать случайных «зацикливаний» в структуре Ethernet сети.

5.2. Создание и настройка интерфейса bridge0

Следующий этап это собственно создание моста. В том же системном руководстве (**man bridge**) читаем дальше и видим, что для того чтобы в системе появился сетевой мост, необходимо добавить следующие строки в **/etc/rc.conf**:

```
EXAMPLES
     The following when placed in the file /etc/rc.conf will cause a bridge
     called “bridge0” to be created, and will add the interfaces “wlan0” and
     “fxp0” to the bridge, and then enable packet forwarding.  Such a
     configuration could be used to implement a simple 802.11-to-Ethernet
     bridge (assuming the 802.11 interface is in ad-hoc mode).

           cloned_interfaces="bridge0"

       ...

     For the bridge to forward packets, all member interfaces and the bridge
     need to be up.  The above example would also require:

       ...

           ifconfig_bridge0="addm em0 addm em1 DHCP"
           ifconfig_em0="up"
           ifconfig_em1="up"
```

Приведенный пример создает новый интерфейс **bridge0** и добавляет в него интерфейсы wlan0, fxp0, em0 и em1. В нашем случае эти добавочные (входящие) интерфейсы следует заменить на **re0** и **re1**. Параметр **DHCP** в настройке **bridge0** указывает на то, что мост будет работать в том числе и как обычный сетевой интерфейс, IP адрес на него будет запрашиваться у DHCP сервера. Это очень полезное свойство моста позволит нам использовать его как подлежащий транспорт для утилиты **mpd5** для организации обратного туннеля. Суммируя всё выше сказанное получим следующий список команд для настройки сетевого моста:

```
sniffer@sniffer:~ % sudo sysrc cloned_interfaces="bridge0"
cloned_interfaces:  -> bridge0

sniffer@sniffer:~ % sudo sysrc ifconfig_bridge0="addm re0 addm re1 SYNCDHCP"
ifconfig_bridge0:  -> addm re0 addm re1 SYNCDHCP

sniffer@sniffer:~ % sudo sysrc ifconfig_re0="up"
```

```
ifconfig_re0:  -> up

sniffer@sniffer:~ % sudo sysrc ifconfig_re1="up"
ifconfig_re1:  -> up
```

И, вобщем-то, всё. Теперь нам нужно перезагрузить операционную систему, чтобы загрузились два новых драйвера. Сделаем это командой **reboot**:

```
sniffer@sniffer:~ % sudo reboot

Connection to 192.168.171.151 closed.
```

Сразу после ввода этой команды соединение по SSH будет разорвано и машина уйдет на перезагрузку. Через минуту мы снова сможем подключиться к ней командой **ssh**.

Подключимся к системе и посмотрим как выглядит созданный нами сетевой интерфейс **bridge0**, еще раз воспользовавшись утилитой **ifconfig**:

```
sniffer@sniffer:~ % ifconfig bridge0
bridge0: flags=1008843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST,LOWER_UP> metric 0 mtu 1500
        options=0
        ether 58:9c:fc:10:ff:d7
        id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
        root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
        member: re1 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
                ifmaxaddr 0 port 2 priority 128 path cost 55
        member: re0 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
                ifmaxaddr 0 port 1 priority 128 path cost 55
        groups: bridge
        nd6 options=9<PERFORMNUD,IFDISABLED>
```

Здесь мы видим, что **bridge0** содержит два других интерфейса-участника: **member: re0** и **member: re1**.

5.3. Проверка работоспособности сетевого моста

Теперь нам нужно проверить как работает сетевой мост. Для этого нам нужно подключить наш сниффер в разрез какого-то другого (исследуемого) хоста или сетевого устройства, исследуемое устройство далее будем называть DUT от англ. «Device Under TEST». Самый простой способ который я смог придумать — это подключить через сниффер конторский сетевой принтер (МФУ) HP LaserJet Pro 500.

Я взял один дополнительный UTP патч-корд и соединил им МФУ со сниффером подключив в первый порт Etherner (совершенно без различно в какой именно порт **re0** или **re1** он будет подключен). Второй патч-корд, которым был до этого подключен МФУ к сети, я соединил второй Ethernet порт сниффера с сетью, т. е. с конторским Ethernet коммутатором. Через пару секунд светодиоды на портах сниффера весело и ритмично замигали, а утилита **ifconfig** теперь показывает следующее:

```
sniffer@sniffer:~ % ifconfig re0
re0: flags=1008943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST,LOWER_UP> metric 0 mtu
1500
        options=8209b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,WOL_MAGIC,LINKSTATE>
        ether 68:1d:ef:47:de:0a
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>

sniffer@sniffer:~ % ifconfig re1
re1: flags=1008943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST,LOWER_UP> metric 0 mtu
1500
        options=8209b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,WOL_MAGIC,LINKSTATE>
```

```
        ether 68:1d:ef:47:de:0b
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Из вывода команды мы видим, что оба интерфейса **re0** и **re1** перешли в рабочее состояние: на обоих интерфейса высвечивается **status: active** и к списку флагов добавился флаг **UP**.

Попробуем подслушать ICMP трафик на мостовом соединении с помощью системной утилиты **tcpdump**. Она кстати, в ходит в стандартную поставку ОС FreeBSD, т. е. ничего устанавливать не требуется.

Введем команду:

```
sniffer@sniffer:~ % sudo tcpdump -i bridge0 icmp
Password: reffins
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on bridge0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Утилита **tcpdump** запущена и слушает трафик сети на мостовом соединении, выбирая из него только ICMP пакеты, о чем говорит строка **icmp** задающая правило фильтрации (о правилах мы поговорим более подробно чуть дальше).

Далее я открываю новое окно терминала на рабочем компьютере и ввожу команду **ping**:

```
rz@butterfly:~ % ping 192.168.169.14
PING 192.168.169.14 (192.168.169.14): 56 data bytes
64 bytes from 192.168.169.14: icmp_seq=0 ttl=254 time=4.501 ms
64 bytes from 192.168.169.14: icmp_seq=1 ttl=254 time=3.283 ms
64 bytes from 192.168.169.14: icmp_seq=2 ttl=254 time=5.427 ms
^C
--- 192.168.169.14 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 3.283/4.404/5.427/0.878 ms
```

Команда **ping** отправляет ICMP запросы на хост с указанным IP адресом. В данном случае 192.168.169.14 это адрес конторского МФУ. Я дожидаюсь отправки трех пакетов и завершаю работу **ping** нажатием **Ctrl-C**.

Переключившись в окно терминала подключенного по SSH к снифферу я тут же наблюдаю следующие сообщения от утилиты **tcpdump**:

```
20:20:36.309894 IP 192.168.176.155 > 192.168.169.14: ICMP echo request, id 63017, seq 0, length 64
20:20:36.310397 IP 192.168.169.14 > 192.168.176.155: ICMP echo reply, id 63017, seq 0, length 64
20:20:37.373156 IP 192.168.176.155 > 192.168.169.14: ICMP echo request, id 63017, seq 1, length 64
20:20:37.373573 IP 192.168.169.14 > 192.168.176.155: ICMP echo reply, id 63017, seq 1, length 64
20:20:38.437588 IP 192.168.176.155 > 192.168.169.14: ICMP echo request, id 63017, seq 2, length 64
20:20:38.437957 IP 192.168.169.14 > 192.168.176.155: ICMP echo reply, id 63017, seq 2, length 64
```

Для завершения работы **tcpdump** необходимо нажать **Ctrl-C**. Я нажимаю и попадаю обратно в оболочку **tcsh**, при этом **tcpdump** перед завершением выдаст немного накопленной статистики о происходившем процессе:

```
^C
6 packets captured
365 packets received by filter
0 packets dropped by kernel
sniffer@sniffer:~ %
```

Из сообщений **tcpdump** мы видим, что через интерфейс за это время прошло много пакетов, но правилам фильтра соответствовало всего 6 пакетов. Информацию об этих шести пакетах утилита **tcpdump** вывела ранее, это три пары пакетов: типа **ICMP echo request** от 92.168.176.155 > 192.168.169.14 (к МФУ от моего рабочего ПК), и типа **ICMP echo reply** от 192.168.169.14 > 192.168.176.155 (ответ от МФУ). Очевидно, что **tcpdump** теперь может подсматривать любой трафик проходящий через мост, как в сторону МФУ, так и от него в сторону сети. Далее мы чуть более подробно изучим работу **tcpdump** и некоторых других утилит для анализа трафика, а также ознакомимся с правилами фильтрации предоставляемыми встроенным в ОС средством под названием Berkeley Packet Filter (BPF).

5.4. Схемы включения сниффера

Перед тем как перейти к описанию способов анализа трафика, рассмотрим несколько вариантов включения сниффера. Вообще, способов включения сниффера может быть очень много, зависят они от структуры эксплуатируемой сети и от способа организации управляющего канала для доступа к снифферу. Ниже на рис. 8а, 8б, 8в и 8г приведены четыре способа включения сниффера в сеть, все они достаточно простые несмотря на некоторую загроможденность изображений. Рассмотрим все четыре варианта.

5.4.1. Включение сниффера в локальной сети, управление по WiFi

Самая простая схема включения сниффера (рис. 8а) состоит в том, чтобы подключить его в разрез между исследуемым устройством (DUT) и остальной сетью. Это тот способ, который был описан чуть выше в эксперимента с МФУ. Канал управления сниффером в этом случае можно организовать через локальную беспроводную сеть WiFi. Логический путь канала управления обозначен на схеме красной линией.

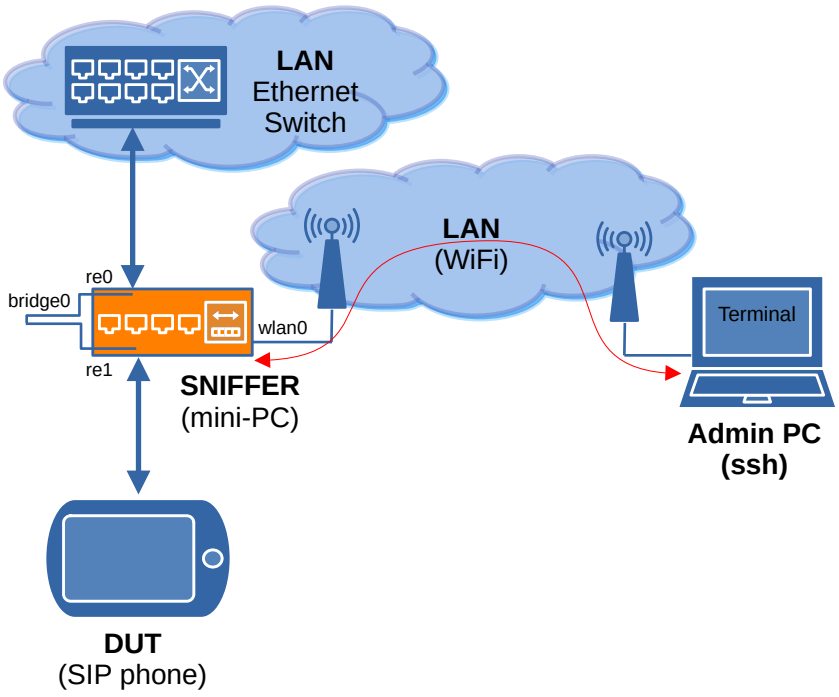

Рис.8а. Подключения сниффера в локальной сети. Управления через WiFi.

Чтобы подключиться к снифферу потребуется рабочий ПК включенный в ту же сеть WiFi, что и сниффер, а также знание IP адреса который выдается снифферу DHCP сервером. Если это домашняя сеть или сеть предприятия, то выяснить IP сниффера не составит большого труда — достаточно заглянуть в лог DHCP сервера или просмотреть список подключенных MAC адресов на домашнем WiFi маршрутизаторе и отыскать там MAC адрес WiFi порта сниффера (заблаговременно введенная команда **ifconfig wlan0 ether** на сниффере покажет его MAC). Подключение к снифферу по SSH осуществляется командой **ssh sniffer@x.x.x.x**, где x.x.x.x — локальный IP адрес сниффера.

5.4.2. Включение сниффера с управлением через PPTP/L2TP туннель

На рис. 8б приведена, по сути, та же схема включения сниффера между DUT и коммутатором локальной сети, но канал управления организован через глобальную сеть Интернет посредства туннеля до PPTP/L2TP сервера. Для того, чтобы у сниффера была возможность поднять туннель до заданного сервера, необходимо чтобы внутри локальной сети а) снифферу выдавался локальный IP адрес по DHCP, и б) чтобы с этого IP был доступ в Интернет. При выполнении этих условий, сниффер при загрузке поднимет внутренний сетевой интерфейс **bridge0** и получит для него IP адрес через DHCP. Далее вступит в действие утилита mpd5 которая автоматически поднимет PPTP/L2TP туннель получив для него дугой IP доступный для подключения снаружи. Для доступа к снифферу по SSH в этом случае используем этот второй IP. Важно чтобы провайдер, обеспечивающий PPTP/L2TP сервис, выдавал глобально маршрутизируемый IP адрес, иначе добраться до сниффера не получиться.

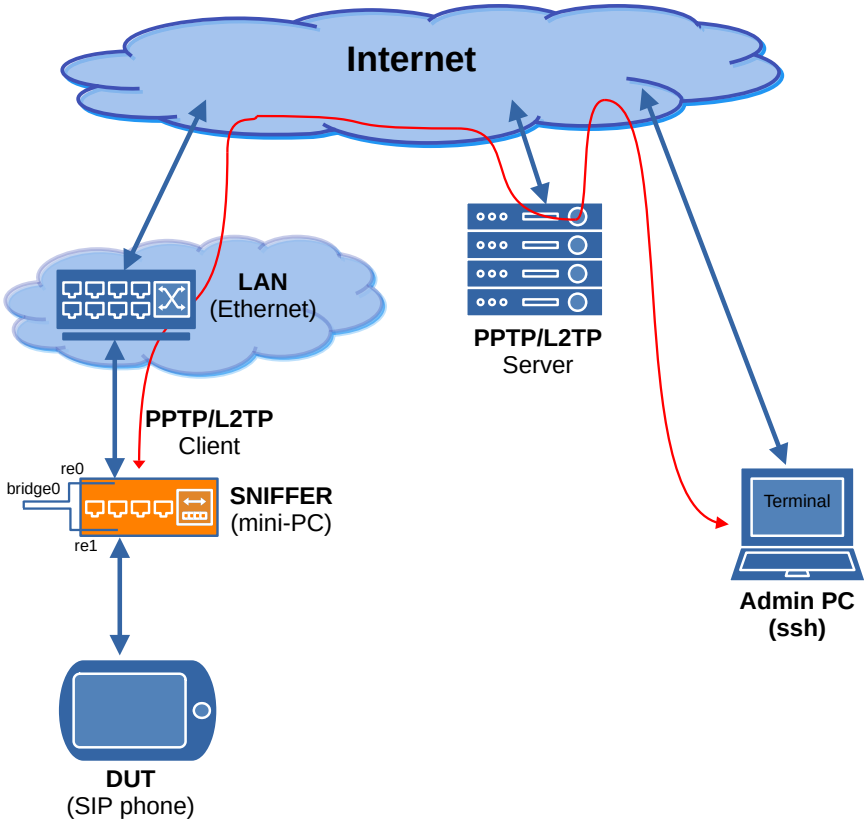

Рис.8б. Подключения сниффера в локальной сети.
Управления через глобальную сеть Internet с помощью туннеля PPTP или L2TP.

5.4.3. Включение сниффера до локальной сети, управление по PPPoE

Часто бывает так, что сеть пользователя имеет стык с глобальной сетью (с сетью провайдера услуг доступа к Интернет) через порт Ethernet. В этом случает сниффер можно включить между локальной сетью и сетью Интернет-провайдера, как показано на рис. 8в. Чтобы организовать канал управления сниффером, потребуется отдельный аккаунт на PPPoE сервера интернет-провайдера обеспечивающего связь.

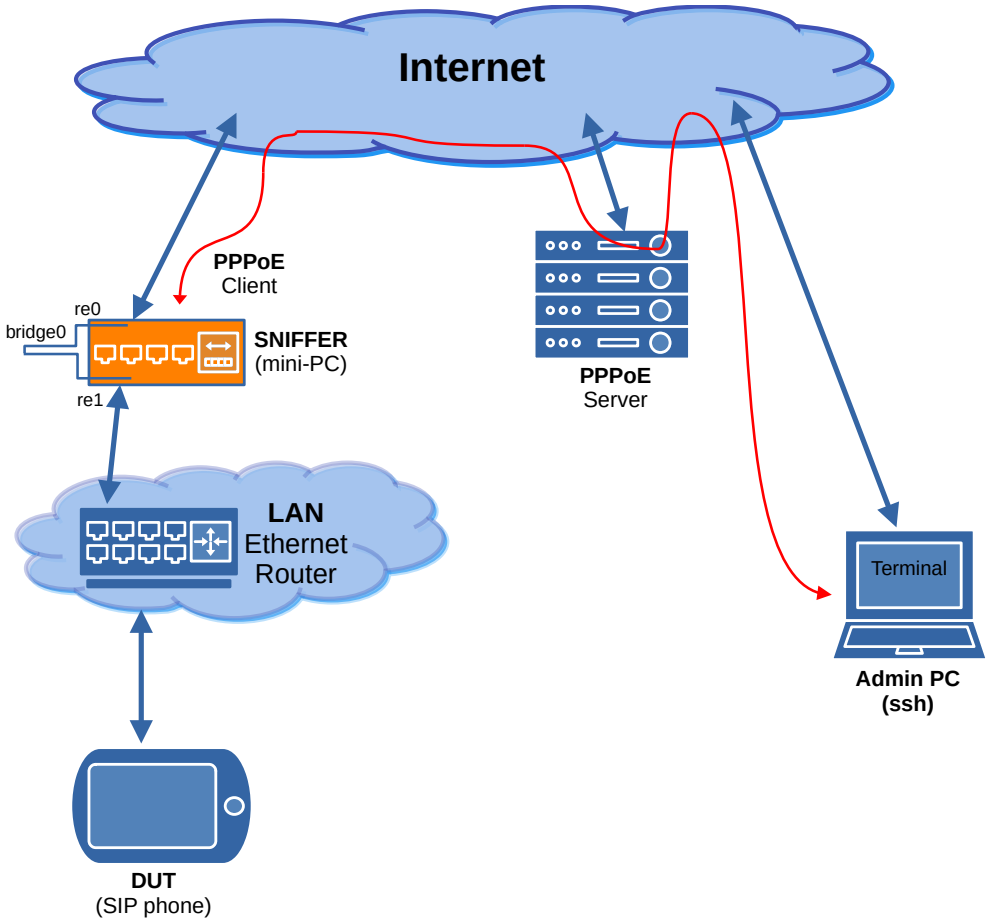

Рис.8г. Подключения сниффера между локальной сетью и интернет-провайдером. Управления через глобальную сеть Internet с помощью туннеля PPPoE.

Аналогично предыдущему варианту, сниффер автоматически поднимет туннель через PPPoE соединение с сервером провайдера и обеспечит себе глобально маршрутизируемый IP адрес для установки SSH соединения с ним.

Важно, что в такой схеме включения через сниффер будет проходить весь трафик сети пользователя, что может создать проблемы с его анализом или с большой нагрузкой на сниффер. Для анализа в такой схеме используйте правила ограничивающие трафик только IP адресом исследуемого устройства. Так же важно отметить, что локальный трафик создаваемый DUT не будет проходить через сниффер и выпадет из анализа, т. е. исследовать в таком случае возможно только глобальный трафик между DUT и хостами в сети Интернет.

5.4.4. Включение сниффера в домашней (GPON) сети, управление по PPPoE

Ну и последний вариант схемы включения, приведенный на рис. 8г, позволяет включить сниффер в домашней сети имеющей соединение с сетью провайдера по оптическому каналу (обычно это GPON). Сниффер устанавливается в разрез между ONU (оптическим оконечным устройством) и домашним маршрутизатором. При таком включении снифферу будет доступна только та часть трафика пользователя, которая выходит в и приходит из глобальной сети.

Канал управления организуется также через PPPoE сервер провайдера. Но, в общем-то, никто не запрещает подключить сниффер к локальному (домашнему) WiFi, а через него, при необходимости иметь удаленное управление, поднять PPTP/L2TP туннель к публичному серверу, как в предыдущих двух варианта.

Если Вы решили исследовать свою домашнюю сеть, Вам достаточно подключить сниффер как показано на этой схеме и управлять им локально, через свою домашнюю сеть WiFi, подключаясь по SSH напрямую.

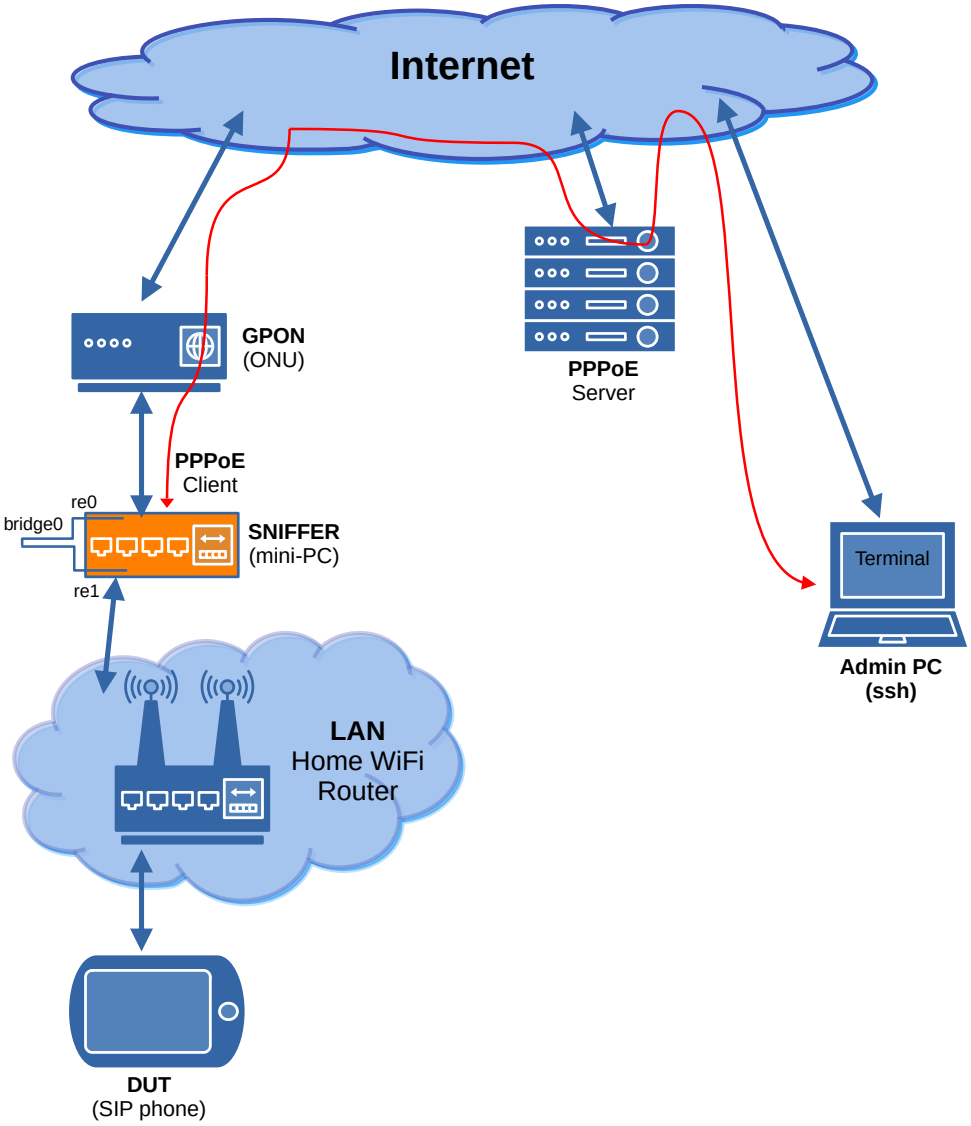

Рис.8д. Подключения сниффера между ONU и абонентским (домашним) маршрутизатором. Управления через глобальную сеть Internet с помощью туннеля PPPoE.

6. Анализ сетевого трафика

Механизм фильтрации сетевого трафика предназначен для того, чтобы пользователю (приложению) можно было выбрать (захватить) и проанализировать только интересующий тип трафика, трафик между определенными хостами или даже портами внутри хоста. Механизм фильтрации BPF встроен в ОС FreeBSD, что называется, из коробки. А это означает, что для него есть страница системного руководства.

6.1. Настройка прав доступа к BPF

Введем команду **man** с параметром **bpf** и кратко ознакомимся с содержимым страницы руководства по BPF:

```
sniffer@sniffer:~ % man bpf
BPF(4)                FreeBSD Kernel Interfaces Manual                BPF(4)

NAME
     bpf – Berkeley Packet Filter

SYNOPSIS
     device bpf

DESCRIPTION
     The Berkeley Packet Filter provides a raw interface to data link layers
     in a protocol independent fashion.  All packets on the network, even
     those destined for other hosts, are accessible through this mechanism.

     The packet filter appears as a character special device, /dev/bpf.  After
     opening the device, the file descriptor must be bound to a specific
     network interface with the BIOCSETIF ioctl.  A given interface can be
     shared by multiple listeners, and the filter underlying each descriptor
     will see an identical packet stream.
```

Из второго абзаца мы узнаем, что доступ к BPF осуществляет через специальный файл-устройство **/dev/bpf**. Посмотрим, кто имеет доступ к этому файлу с помощью утилиты **ls**, введем команду:

```
sniffer@sniffer:~ % ls -al /dev/bpf*
crw-------  1 root wheel 0x27 Jul 10 18:38 /dev/bpf
lrwxr-xr-x  1 root wheel      3 Jul 10 18:20 /dev/bpf0 -> bpf
```

Видно, что файл-устройство **/dev/bpf** и символическая ссылка на него **/dev/bpf0** принадлежат группе **wheel**, при этом у группы нет прав доступа к этому файлу (в соответствующем поле стоит прочерк: **--**). Это означает, что только пользователь с привилегиями администратора может получать доступ к BPF, то есть нам всегда придется использовать утилиту **sudo**. При частом использовании это не очень удобно. Поэтому сменим принадлежность и права доступа к **/dev/bpf** введя следующие команды:

```
sniffer@sniffer:~ % sudo chmod g+w /dev/bpf
sniffer@sniffer:~ % sudo chgrp network /dev/bpf
```

Посмотри как изменились атрибуты и принадлежность этого файла:

```
sniffer@sniffer:~ % ls -al /dev/bpf
crw-rw----  1 root network 0x27 Jul 10 18:38 /dev/bpf
```

Теперь любой пользователь в группе **network** может читать и писать (**rw**) в этот файл-устройство, а значит для доступа к BPF нам больше не требуется использовать **sudo**

(напомню, что пользователь **sniffer**, под которым мы подключаемся к системе, входит в группу **network**).

Тут есть небольшой нюанс. После перезагрузки операционной системы, права доступа на этот файл вернутся к исходному виду, а это совсем не то, чего мы ожидаем. Существует системный механизм **devfs** отслеживающий и изменяющий права доступа к системным файлам, с помощью него мы может устанавливать права доступа на отдельные устройства так, как нам этого требуется. Добавим, с помощью редактора **ee**, в конфигурационный файл **/etc/devfs.conf** следующие два правила для **bpf**:

```
sniffer@sniffer:~ % sudo ee /etc/devfs.conf
```

Скопируем следующие строки в буфер обмена:

```
own  bpf* root:network
perm bpf* 0660
```

и поместим их в файл. Выйдем из редактора с сохранением изменений. Теперь в момент загрузки операционной системы **devfs** будет устанавливать требуемые нам права на доступ к BPF и мы сможем использовать различные утилиты для анализа трафика без вызова утилиты **sudo**.

6.2. Правила фильтрации трафика в BPF

Сразу скажу, что тема BPF (Berkeley Packet Filter) достаточно обширная, уходящая корнями в 1980-е годы к машинам и системам производства Digital Equipment Corporation, и может потребовать написания отдельной многостраничной публикации. Строго говоря, BPF это виртуальная машина внутри ядра операционной системы, со своей системой команд (своими инструкциями) и своими форматами данных. Из этих инструкций, аналогично тому как складывается программа для центрального процессора из машинных кодов, составляют программы для фильтрующего механизма (Filter Machine). Исполняясь внутри ядра операционной системы, BPF программы могут не только анализировать и фильтровать сетевой трафик, но и осуществлять его модификацию (подставлять и заменять данные в пакетах «на лету»), что делает BPF очень мощным и важным механизмом при организации систем безопасности, обнаружения и автоматического пассивного и активного противодействия вторжениям. BPF позволяет строить очень сложные межсетевые экраны (firewall-ы), а операционные системы FreeBSD и OpenBSD, с BPF на борту, широко используются в промышленных изделия и системах сетевой безопасности. Но сейчас речь не об этом. В рамках этой статьи я по попытаюсь кратко изложить основные моменты для анализа трафика и привести пару рабочих примеров.

Если продолжить чтение руководства по BPF (**man bpf**), то мы увидим, что основная его часть описывает программные механизмы (API) на языке Си и принципы создания BPF программ из пользовательских и системных приложений. Это не совсем то, что нам требуется, так как мы не собираемся разрабатывать своих программ, а хотим воспользоваться уже существующими. Имеется специальная библиотека **libpcap** (PCAP) которая используется многими утилитами для захвата, анализа и фильтрации сетевого трафика. Эта библиотека предоставляет приложениям унифицированный интерфейс к BPF на основе текстовых правил понятных конечному пользователю, что позволяет использовать BPF без дополнительного программирования. С описанием синтаксиса языка для создания правил фильтра BPF можно ознакомиться на отдельной странице системного руководства по команде **man pcap-filter**. Давайте кратко ознакомимся хотя бы с первой страницей этого руководства:

```
sniffer@sniffer:~ % man pcap-filter

PCAP-FILTER(7)     FreeBSD Miscellaneous Information Manual     PCAP-FILTER(7)

NAME
       pcap-filter - packet filter syntax

DESCRIPTION
       pcap_compile(3) is used to compile a string into a filter program.  The
       resulting filter program can then be applied to some stream of packets
       to determine which packets will be supplied to pcap_loop(3),
       pcap_dispatch(3), pcap_next(3), or pcap_next_ex(3).

       The filter expression consists of one or more primitives.  Primitives
       usually consist of an id (name or number) preceded by one or more
       qualifiers.  There are three different kinds of qualifier:

       type   type qualifiers say what kind of thing the id name or number
              refers to.  Possible types are host, net, port and portrange.
              E.g., `host foo', `net 128.3', `port 20', `portrange 6000-6008'.
              If there is no type qualifier, host is assumed.

       dir    dir qualifiers specify a particular transfer direction to and/or
              from id.  Possible directions are src, dst, src or dst, src and
              dst, ra, ta, addr1, addr2, addr3, and addr4.  E.g., `src foo',
              `dst net 128.3', `src or dst port ftp-data'.  If there is no dir
              qualifier, `src or dst' is assumed.  The ra, ta, addr1, addr2,
              addr3, and addr4 qualifiers are only valid for IEEE 802.11
              Wireless LAN link layers.

       proto  proto qualifiers restrict the match to a particular protocol.
              Possible protocols are: ether, fddi, tr, wlan, ip, ip6, arp,
              rarp, decnet, sctp, tcp and udp.  E.g., `ether src foo', `arp
              net 128.3', `tcp port 21', `udp portrange 7000-7009', `wlan
              addr2 0:2:3:4:5:6'.  If there is no proto qualifier, all
              protocols consistent with the type are assumed.  E.g., `src foo'
              means `(ip6 or ip or arp or rarp) src foo', `net bar' means `(ip
              or arp or rarp) net bar' and `port 53' means `(tcp or udp or
              sctp) port 53' (note that these examples use invalid syntax to
              illustrate the principle).
```

И немного далее:

```
       Primitives may be combined using:

              A parenthesized group of primitives and operators.

              Negation (`!' or `not').

              Concatenation (`&&' or `and').

              Alternation (`||' or `or').

       Negation has the highest precedence.  Alternation and concatenation
       have equal precedence and associate left to right.

       If an identifier is given without a keyword, the most recent keyword is
       assumed.  For example,
            not host vs and ace
       is short for
            not host vs and host ace
       which should not be confused with
            not (host vs or ace)
```

В этой цитате из руководства содержится максимум знаний которых нам требуется для составления весьма сложных фильтров. Если коротко изложить сказанное в руководстве, то правила фильтрации выглядят следующим образом:

1. Примитивное правило состоит из трех честей: **type dir proto** где:

- **type** — задает тип сущности определяемой далее в правиле, это может быть **host**, **net**, **port** или **portrange**.
- **dir** — определяет какое из полей пакета будет анализироваться: **src**, **dst**, **addr1**, **addr2**, и т. д.
- **proto** — указывает пакеты относящиеся к какому протоколу будут соответствовать этому правилу: **ether**, **fddi**, **tr**, **wlan**, **vlan**, **mpls**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **sctp**, **icmp**, **tcp, udp** и т. д. (их очень много).

2. Любая из частей правила может отсутствовать, что равносильно выражению («any» - любой). То есть если отсутствует **proto**, то будут анализироваться пакеты всех протоколов.
3. Примитивные правила могут объединяться в более сложные правила с помощью логических операций **or**, **and** и **not**, а так же вложенной системы круглых скобок. Логические операции можно выражать символами: **||**, **&&** или **!**

Для ясности рассмотрим несколько простых примеров:

1. Правило из одного слова **icmp** создаст фильтр который описывает только Internet Control and Management Protocol (ICMP), то есть пакеты создаваемые запросами команды **ping** и ответы на них. Это правило мы уже использовали для тестирования сетевого моста в главе *5.3*.

2. Правило: **udp 5060** создаст фильтр который покроет UDP трафик только для портов 5060 (в обе стороны). Это простейшее правило для выделения SIP пакетов из общего потока сетевого трафика.

3. Правило: **not port 22** создаст фильтр, который исключит из захватываемого трафика все пакеты (TCP и UDP) в назначении или источнике которых присутствует порт номер **22**. Это простейшее правило для того, чтобы вычесть (удалить из анализа) трафик создаваемый протоколом удаленного доступа SSH. Очень полезное правило, применимо когда требуется посмотреть на весь сетевой трафик, но трафик удаленного управления хостом (SSH) может забить вывод гигантским потоком бесполезной информации. Здесь это правило и приходит на помощь.

4. Правило: **dst host 192.168.168.14** создаст фильтр выделяющий поток пакетов в поле IP адреса назначения которых указан заданный хост. С помощью этого правила можно подсмотреть трафик направляющийся в сторону хоста **192.168.168.14**.

5. Правило: **vlan 2** выделит все Ethernet пакеты типа **0x8100** в которых тэг VLAN равен **2**.

6. Чуть более сложное правило: **vlan 4 and dst host 192.168.168.14 and not port 22** выделить трафик для VLAN=2 и только в сторону хоста **192.168.168.14**, удалит из выборки трафик протокола SSH (порт **22**) чтобы не мешался.

6.3. Правила фильтрации трафика в BPF с использованием DPI

В BPF существуют механизмы для того, чтобы не только рассматривать заголовки пакетов, но и заглянуть поглубже в данные пакета (в payload). Такие правила глубокого

анализа пакетов принято называть Deep Packet Inspection (DPI). Ниже приведены пара примеров DPI правил:

1. Правило: **'udp[8:4] = 0x5349502f && udp[12:4] = 0x322e3020'** позволит выбрать из потока только UDP пакеты содержащие в самом начале блока полезных данных (в начале payload) строку символов: “**SIP/2.0** “ (включая пробел), не зависимо от того с каким номером порта происходит обмен. Пакет с таким содержимым это ответы на запросы протокола SIP/2.0. Такое правило может быть полезно в том случае, когда проприетарное приложение пытается «спрятать» обмен по SIP протоколу выбирая случайный номер UDP порта.

2. Более сложное правило с обратной связью: **'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x5353482d'** позволяет выявить все попытки установить соединение по протоколу SSH независимо от того на каком TCP порту располагается этот сервис. Такое правило может быть очень полезно для обнаружения сканирования портов SSH вредоносным ПО или попытками вторжения.

Как это работает, скорее всего уже стало понятно. Первое правило выбирает из UDP пакета два значения по 4 байта, представляет их в виде числа и сравнивает с заданными константами. Цифра 8 в правиле **udp[8:4]** задает смещение в 8 байт относительно начала UDP заголовка, а цифра 4 указывает на то, что нужно взять для сравнения четыре байта. Константа **0x5349502f** представляет собой строку текста **"SIP/"** в ASCII коде, а **0x322e3020** — строку текста "**2.0 "** вместе с пробелом.

Второе правило гораздо сложнее. Это правило нацелено на анализ содержимого полезной нагрузки TCP пакетов. С TCP пакетами есть некоторая сложность — у них заголовок имеет переменную длину, а значит добраться до полезной нагрузки просто указав смещение — не получится. Для таких случаев можно воспользоваться фичей BPF, позволяющей вычислить значение переменной исходя из данных и использовать её значение как смещение для получения другого значения которое уже будет сравниваться с константой. В приведенном выше примере кусочек правила **((tcp[12:1] & 0xf0) >> 2)** как раз вычисляет длину заголовка, то есть получает смещения нулевого байта полезной нагрузки. Далее это значение подставляется как индекс (смещение) и выбирается другое четырехбайтовое значение для сверки на совпадение с константой **0x5353482d**, аналогично случаю с UDP.

В целом, я думаю идея понятна — зная какое-то число или строку символов, которая обязана присутствовать в пакете, и место её расположения, мы может выбрать этот пакет для анализа. Далее мы попробуем применить эти правила в действии на живой сети.

Правила DPI для анализа трафика создают очень большую нагрузку на вычислительную систему, особенно при интенсивном потоке трафика. Используя несколько таких правил при средненькой нагрузке на сеть можно легко «завесить» систему на снифферe или даже отправить ядро в panic, поэтому действовать надо осторожно — желательно добавлять исключения лишнего (заведомо не интересующего нас) трафика с помощью примитивных правил типа **src host**, **tcp port** и т.д. Я думаю, теперь понятно зачем мы проводили тесты производительности Coremark и STREAM ? ;-)

6.4. Использования BPF фильтров в `tcpdump' и `tshark'

Настало время познакомиться с еще одной мощной утилитой анализа сетевого трафика - **WireShark**. Эта утилита наверняка знаком многим сетевым инженерам и админам, в том числе в среде Windows. WireShark имеет удобный графически интерфейс позволяющий не только захватывать (записывать) и отображать отдельные пакеты, но и производить глубокий анализ записанного трафика с несколькими проходами. WireShark умеет автоматически разделять весь трафик на ассоциированные потоки между конечными точками (парами host:port), позволяет раскрутить TCP цепочки и вынимать из всего объема пакетов последовательный поток передаваемых полезных данных. Иными словами WireShark умеет подсматривать и извлекать из потока данные передаваемые пользовательским приложением. В WireShark имеются гибкие средства поиска по записанному объему данных и фильтрации на основе BPF. На рис. 9 ниже представлено окно графического интерфейса WireShark в среде ОС Linux.

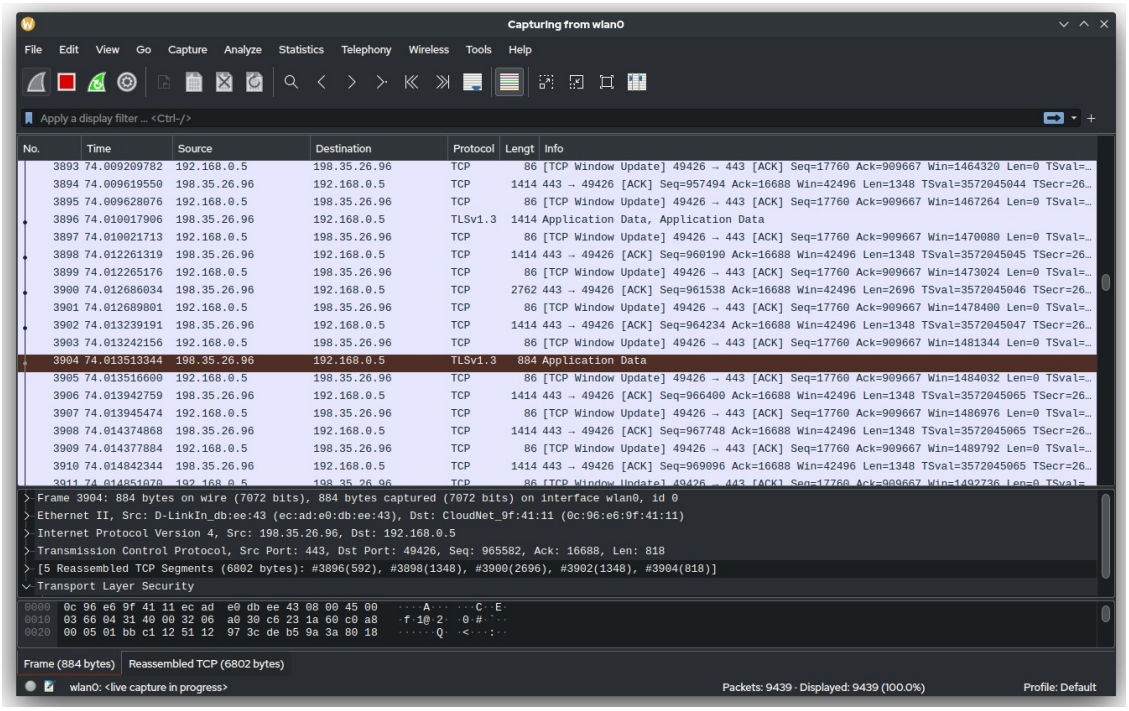

Рис. 9. Окно графического интерфейса WireShark в ОС Linux.

Однако, немногие сетевые инженеры знают про существование «консольного» варианта WireShark, или точнее, варианта этой же утилиты предназначенного для текстового терминала. Называется эта утилита **tshark**. Она входит в тот же пакет программ с общим названием WireShark и использует тот же самый бэкенд, что и её графический собрат.

Утилита **tshark** имеет богатый набор опций и параметров командной строки, но общий подход у неё такой же как у утилиты **tcpdump** — за набором опций следует набор правил BPF, за одним исключением — синтаксис этих правил гораздо богаче. Например, в **tshark** можно указать правило вида '**http.request.method == "GET"**' чтобы получать только пакеты с HTTP запросами. Такое правило тоже относится к классу DPI и внутри утилиты оно представляется в виде комбинации более примитивных правил BPF. Пользоваться утилитой **tshark** удобнее вместо утилиты **tcpdump** в том случае, когда требуется выполнить именно глубокий анализ передаваемой по сети полезной нагрузки.

У **tshark** имеется своя страница системного руководства которую можно вызвать командой **man tshark**. Предлагаю ознакомимся с ней:

```
sniffer@sniffer:~ % man tshark

TSHARK(1)                                                       TSHARK(1)

NAME
       tshark - Dump and analyze network traffic

SYNOPSIS
       tshark [ -i <capture interface>|- ] [ -f <capture filter> ] [ -2 ]
       [ -r <infile> ] [ -w <outfile>|- ] [ options ] [ <filter> ]

       tshark -G [ <report type> ] [ --elastic-mapping-filter <protocols> ]
       [ -C <profile> ]

       tshark -h|--help

       tshark -v|--version

DESCRIPTION
       TShark is a network protocol analyzer.  It lets you capture packet data
       from a live network, or read packets from a previously saved capture
       file, either printing a decoded form of those packets to the standard
       output or writing the packets to a file.  TShark's native capture file
       format is pcapng format, which is also the format used by Wireshark and
       various other tools.

       Without any options set, TShark will work much like tcpdump.  It will
       use the pcap library to capture traffic from the first available
       network interface and displays a summary line on the standard output
       for each received packet.

       ...
```

В руководстве сказано, что **tshark** (разработчики предпочитают писать название с заглавных - **TShark**) во многих случаях ведет себя подобно утилите **tcpdump** и это очень удобно, так как многие администраторы Unix систем привыкли к утилите **tcpdump** и её набору параметров. **tshark** в плане именования опций и составления правил очень похож на **tcpdump**.

Перейдем от слов к делу — опробуем приведенные в предыдущей главе DPI правила на обеих утилитах. Для этого я подключу сниффер в разрез между IP-видеодомофоном и сетью, точно таким же способом как это было описано в главе 5.3 с пропусканием трафика конторского МФУ.

*Следует отметить, что при указании сложных правил в командной строке их следует обрамлять одинарными кавычками, иначе оболочка **tcsh** может интерпретировать символы логических операций и скобок по своему усмотрению и мы можем получить совсем не тот результат.*

6.4.1. Захват SIP пакетов с помощью утилиты `tcpdump’

Вызовем утилиту **tcpdump** и передадим ей следующее BPF правило для глубокого анализа трафика:

```
sniffer@sniffer:~ % tcpdump -n -i bridge0 -s 1500 -X -vv 'udp[8:4] = 0x5349502f && udp[12:4]
= 0x322e3020'
```

Через несколько секунд мы увидим первые пакеты обмена IP-домофона с SIP прокси-сервером. Напомню, что это сложное правило анализирует содержимое полезной нагрузки

UDP пакетов с целью определить и захватить пакеты SIP ответов (к нашему удобству они содержат строку «SIP/2.0» в самом начале payload). Это правило не реагирует на SIP запросы! Один из таких «пойманных» ответов будет выглядеть следующим образом:

```
tcpdump: listening on bridge0, link-type EN10MB (Ethernet), snapshot length 1500 bytes
00:26:23.478943 IP (tos 0x68, ttl 254, id 375, offset 0, flags [none], proto UDP (17), length 412)
    72.243.211.106.53520 > 192.168.168.139.5060: [udp sum ok] SIP, length: 384
        SIP/2.0 100 Trying
        Via: SIP/2.0/UDP 192.168.168.139:5060;branch=z9hG4bK-gdklpgig931861861291050861291.b;rport
        0x0000:  4568 019c 0177 0000 fe11 92e1 4df2 6f6a  Eh...w......M.oj
        0x0010:  c0a8 a88b d110 13c4 0188 5447 5349 502f  ..........TGSIP/
        0x0020:  322e 3020 3130 3020 5472 7969 6e67 0d0a  2.0.100.Trying..
        0x0030:  5669 613a 2053 4950 2f32 2e30 2f55 4450  Via:.SIP/2.0/UDP
        0x0040:  2031 3932 2e31 3638 2e31 3638 2e31 3339  .192.168.168.139
        0x0050:  3a35 3036 303b 6272 616e 6368 3d7a 3968  :5060;branch=z9h
        0x0060:  4734 624b 2d67 646b 6c70 6769 6739 3331  G4bK-gdklpgig931
        0x0070:  3836 3138 3631 3239 3130 3530 3836 3132  8618612910508612
        0x0080:  3931 2e62 3b72 706f 7274 0d0a 4672 6f6d  91.b;rport..From
        0x0090:  3a20 22d0 94d0 bed0 bcd0 bed1 84d0 bed0  :."............
        0x00a0:  bd22 203c 7369 703a 3230 3140 6661 626d  ."·.<sip:201@exam
        0x00b0:  6963 726f 2e72 753e 3b74 6167 3d46 726f  ples.xx>;tag=Fro
        0x00c0:  6d54 6167 2d67 646b 6c70 6769 6739 3331  mTag-gdklpgig931
        0x00d0:  3836 3138 3631 3239 3130 3530 3836 3132  8618612910508612
        0x00e0:  3931 2e62 0d0a 546f 3a20 3c73 6970 3a32  91.b..To:.<sip:2
        0x00f0:  3031 4066 6162 6d69 6372 6f2e 7275 3e0d  01@examples.xx>.
        0x0100:  0a44 6174 653a 2054 6875 2c20 3130 204a  .Date:.Thu,.10.J
        0x0110:  756c 2032 3032 3520 3136 3a33 323a 3235  ul.2025.16:32:25
        0x0120:  2047 4d54 0d0a 4361 6c6c 2d49 443a 2067  .GMT..Call-ID:.g
        0x0130:  646b 6c70 6769 6739 3331 3836 3138 3631  dklpgig931861861
        0x0140:  3239 3130 3530 3836 3132 3931 2e62 0d0a  291050861291.b..
        0x0150:  5365 7276 6572 3a20 4369 7363 6f2d 5349  Server:.Cisco-SI
        0x0160:  5047 6174 6577 6179 2f49 4f53 2d31 322e  PGateway/IOS-12.
        0x0170:  780d 0a43 5365 713a 2032 3833 2052 4547  x..CSeq:.283.REG
        0x0180:  4953 5445 520d 0a43 6f6e 7465 6e74 2d4c  ISTER..Content-L
        0x0190:  656e 6774 683a 2030 0d0a 0d0a            ength:.0....
```

Обмен по SIP протоколу обычно не шифруется, т. е. передается в чистом текстовом виде (plain text), поэтому по содержимому пакета мы может точно определить, что перед нами ответ «SIP/2.0 100 Trying» от SIP-прокси с адресом 72.243.211.106 на запрос «REGISTER», который предположительно был отправлен мгновением ранее. В тексте пакета жирным выделены те байты (то кусочек блока полезной нагрузки), за который ухватилось правило. Очевидно, что если иметь представление о формате полезной нагрузки, то не сложно составить BPF правило для того, чтобы выхватывать из потока любые интересные для анализа пакеты. Но только если эти данные не были переданы в зашифрованном виде. :)

6.4.2. Захват SIP пакетов с помощью утилиты `tshark’

Проделаем этот же фокус, но уже с помощью утилиты **tshark**. Формат команды очень похож на тот, что мы только что использовали для утилиты **tcpdump**:

```
sniffer@sniffer:~ % tshark -n -i bridge0 -s 1500 -V 'udp[8:4] = 0x5349502f && udp[12:4] =
0x322e3020'
```

При первом же совпадении получим в терминале следующее сообщение от **tshark**:

```
Capturing on 'bridge0'
Frame 1: 426 bytes on wire (3408 bits), 426 bytes captured (3408 bits) on interface bridge0, id 0
    Section number: 1
    Interface id: 0 (bridge0)
        Interface name: bridge0
    Encapsulation type: Ethernet (1)
    Arrival Time: Jul 11, 2025 00:54:19.286236877 +05
    UTC Arrival Time: Jul 10, 2025 19:54:19.286236877 UTC
```

```
    Epoch Arrival Time: 1752177259.286236877
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 426 bytes (3408 bits)
    Capture Length: 426 bytes (3408 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:sip]
Ethernet II, Src: e8:39:35:b1:10:ee, Dst: fe:4a:eb:ea:b4:61
    Destination: fe:4a:eb:ea:b4:61
        .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the
factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: e8:39:35:b1:10:ee
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
    [Stream index: 0]
Internet Protocol Version 4, Src: 72.243.211.106, Dst: 192.168.168.139
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x68 (DSCP: AF31, ECN: Not-ECT)
        0110 10.. = Differentiated Services Codepoint: Assured Forwarding 31 (26)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 412
    Identification: 0x01a9 (425)
    000. .... = Flags: 0x0
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 254
    Protocol: UDP (17)
    Header Checksum: 0x92af [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 72.243.211.106
    Destination Address: 192.168.168.139
    [Stream index: 0]
User Datagram Protocol, Src Port: 53520, Dst Port: 5060
    Source Port: 53520
    Destination Port: 5060
    Length: 392
    Checksum: 0x5e47 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 1]
    [Timestamps]
        [Time since first frame: 0.000000000 seconds]
        [Time since previous frame: 0.000000000 seconds]
    UDP payload (384 bytes)
Session Initiation Protocol (100)
    Status-Line: SIP/2.0 100 Trying
        Status-Code: 100
        [Resent Packet: False]
    Message Header
        Via: SIP/2.0/UDP 192.168.168.139:5060;branch=z9hG4bK-gdklpgig931861861291050861291.b;rport
            Transport: UDP
            Sent-by Address: 192.168.168.139
            Sent-by port: 5060
            Branch: z9hG4bK-gdklpgig931861861291050861291.b
            RPort: rport
        From: "Домофон" <sip:201@samples.xx>;tag=FromTag-gdklpgig931861861291050861291.b
            SIP from display info: "Домофон"
            SIP from address: sip:201@samples.xx
                SIP from address User Part: 201
                SIP from address Host Part: samples.xx
            SIP from tag: FromTag-gdklpgig931861861291050861291.b
        To: <sip:201@samples.xx>
            SIP to address: sip:201@samples.xx
                SIP to address User Part: 201
                SIP to address Host Part: samples.xx
        Date: Thu, 10 Jul 2025 17:00:21 GMT
        Call-ID: gdklpgig931861861291050861291.b
        [Generated Call-ID: gdklpgig931861861291050861291.b]
```

```
Server: Cisco-SIPGateway/IOS-12.x
CSeq: 308 REGISTER
    Sequence Number: 308
    Method: REGISTER
Content-Length: 0
```

Вывод у **tshark** будет гораздо более насыщенным. Для каждого пакета эта утилита выдает следующий анализ:

- анализа Ethernet II (MAC) заголовка;
- анализа Internet Protocol (IP) заголовка;
- анализа User Datagram Protocol (UDP) заголовка;
- и анализ Session Initiation Protocol (SIP) пакета.

Утилита **tshark** попытается проанализировать буквально каждый бит захваченного пакеты и выдать максимум имеющейся у неё информации. Из приведенного выше листинга мы видим, что **tshark** точно определила что это именно SIP пакет содержащий ответ на запрос «REGISTER» и досконально расписала нам содержимое каждого поля.

Стоит отметить, что утилита **tcpdump** тоже выполняет некоторый анализ (парсинг) MAC и IP заголовков, однако анализ SIP она делать не способна. Хотя вывести содержимое SIP пакета в текстовом виде возможно.

6.4.3. Детектирование соединений SSH с помощью утилиты `tcpdump’

Теперь опробуем более сложное (второе) правило и посмотрим как с ним справляются обе утилиты. Напомню, что это правило сначала находит в TCP пакете расположение полезной нагрузки, а затем пытается определить в ней строку текста **SSH_** в самом начале, что является признаком хэндшейка по протколу SSH.

Сначала запустим утилиту **tcpdump** со следующими параметрами:

```
sniffer@sniffer:~ % tcpdump -n -i bridge0 -s 1500 -X -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] =
0x5353482d'
```

Затем я открою еще одно окно терминала на своем рабочем ПК и попробую установить соединение по SSH с IP-домофоном который находится на IP адресе 192.168.168.139 — это мы только что определили из предыдущего опыта. На домофоне предварительно запущен демон **sshd** на порту **22/tcp**, иначе опыт не состоится. :)

Я выполняю следующую команду на рабочем ПК:

```
rz@butterfly:~ % ssh fabmicro@192.168.168.139
```

И тут же наблюдаю в терминале сниффера следующее сообщение от **tcpdump**, содержащее дамп «пойманного» пакета:

```
tcpdump: listening on bridge0, link-type EN10MB (Ethernet), snapshot length 1500 bytes
00:25:10.212128 IP (tos 0x48, ttl 63, id 0, offset 0, flags [DF], proto TCP (6), length 90)
    192.168.176.155.37196 > 192.168.168.139.22: Flags [P.], cksum 0x63d4 (correct), seq
3500301681:3500301719, ack 2008447800, win 257, options [nop,nop,TS val 2915515975 ecr 962763132],
length 38: SSH: SSH-2.0-OpenSSH_9.9 FreeBSD-20250219
        0x0000:  4548 005a 0000 4000 3f06 60de c0a8 b09b  EH.Z..@.?.`.....
        0x0010:  c0a8 a88b 914c 0016 d0a2 5d71 77b6 7b38  .....L....]qw.{8
        0x0020:  8018 0101 63d4 0000 0101 080a adc7 3e47  ....c.........>G
        0x0030:  3962 997c 5353 482d 322e 302d 4f70 656e  9b.|SSH-2.0-Open
```

```
        0x0040:  5353 485f 392e 3920 4672 6565 4253 442d  SSH_9.9.FreeBSD-
        0x0050:  3230 3235 3032 3139 0d0a                 20250219..
```

Мы видим, что **tcpdump** изловил из всего потока трафика TCP пакет по содержанию его полезной нагрузки. Как и в прошлом эксперименте я выделил жирным текстом часть байтов, которые удовлетворили указанному правилу.

Из содержимого пакета видно, что **ssh** клиент входит в состав пакета OpenSSH и работает под управлением ОС FreeBSD, а по строке содержащей дату сборки клиента даже можно предположить какой именно версии. Вот так, подсмотрев всего один пакет можно уже достаточно много узнать о пользователе. :)

6.4.4. Детектирование соединений SSH с помощью утилиты `tshark’

Опробуем это же правило с утилитой **tshark**, введя следующую команду:

```
sniffer@sniffer:~ % tshark -n -i bridge0 -s 1500 -V 'tcp[((tcp[12:1] & 0xf0) >> 2):4] =
0x5353482d'
```

и выполним попытку соединения по SSH с IP-домофоном аналогично предыдущему опыту. В окне терминала сниффера сразу получим длинный листинг с анализом TCP пакета от **tshark**:

```
Capturing on 'bridge0'
Frame 1: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface bridge0, id 0
    Section number: 1
    Interface id: 0 (bridge0)
        Interface name: bridge0
    Encapsulation type: Ethernet (1)
    Arrival Time: Jul 11, 2025 00:36:25.038883854 +05
    UTC Arrival Time: Jul 10, 2025 19:36:25.038883854 UTC
    Epoch Arrival Time: 1752176185.038883854
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 104 bytes (832 bits)
    Capture Length: 104 bytes (832 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:ssh]
Ethernet II, Src: e8:39:35:b1:10:ee, Dst: fe:4a:eb:ea:b4:61
    Destination: fe:4a:eb:ea:b4:61
        .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the
factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: e8:39:35:b1:10:ee
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
    [Stream index: 0]
Internet Protocol Version 4, Src: 192.168.176.155, Dst: 192.168.168.139
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x48 (DSCP: AF21, ECN: Not-ECT)
        0100 10.. = Differentiated Services Codepoint: Assured Forwarding 21 (18)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 90
    Identification: 0x0000 (0)
    010. .... = Flags: 0x2, Don't fragment
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 63
    Protocol: TCP (6)
    Header Checksum: 0x60de [validation disabled]
```

```
    [Header checksum status: Unverified]
    Source Address: 192.168.176.155
    Destination Address: 192.168.168.139
    [Stream index: 0]
Transmission Control Protocol, Src Port: 45404, Dst Port: 22, Seq: 1, Ack: 1, Len: 38
    Source Port: 45404
    Destination Port: 22
    [Stream index: 0]
    [Stream Packet Number: 1]
    [Conversation completeness: Incomplete (0)]
        ..0. .... = RST: Absent
        ...0 .... = FIN: Absent
        .... 0... = Data: Absent
        .... .0.. = ACK: Absent
        .... ..0. = SYN-ACK: Absent
        .... ...0 = SYN: Absent
        [Completeness Flags: [ Null ]]
    [TCP Segment Len: 38]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2043450806
    [Next Sequence Number: 39    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2021031524
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Accurate ECN: Not set
        .... 0... .... = Congestion Window Reduced: Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····AP···]
    Window: 257
    [Calculated window size: 257]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x8b83 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        TCP Option - No-Operation (NOP)
            Kind: No-Operation (1)
        TCP Option - No-Operation (NOP)
            Kind: No-Operation (1)
        TCP Option - Timestamps: TSval 2682966770, TSecr 963437961
            Kind: Time Stamp Option (8)
            Length: 10
            Timestamp value: 2682966770
            Timestamp echo reply: 963437961
    [Timestamps]
        [Time since first frame in this TCP stream: 0.000000000 seconds]
        [Time since previous frame in this TCP stream: 0.000000000 seconds]
    [SEQ/ACK analysis]
        [Bytes in flight: 38]
        [Bytes sent since last PSH flag: 38]
    TCP payload (38 bytes)
SSH Protocol
    Protocol: SSH-2.0-OpenSSH_9.9 FreeBSD-20250219
    [Direction: client-to-server]
```

Аналогично тому, как это было с SIP пакетом, утилита **tshark** выдала детальный анализ пакета, являющегося частью SSH хэндшейка состоящий из четырех разделов:

- анализа Ethernet II (MAC) заголовка;
- анализа Internet Protocol (IP) заголовка;
- анализа Transmission Control Protocol (TCP) заголовка;
- и анализа SSH Protocol (SSH) пакета.

6.4.5. Анализ HTTP запросов: добываем логин и пароль с помощью утилиты `tshark’

Приведу еще один пример демонстрирующий возможности утилиты **tshark** по декодированию потока данных из общего трафика. Предположим, что у нас есть некоторое устройство, тот же IP-домофон, с Web-админкой на борту и нам требуется выяснить текст HTTP запросов и ответов на них, в частности — подсмотреть пароль для прохождения авторизации (причины для этого могут быть чисто технические). Последовательность действий следующая:

Сначала мы запишем весь HTTP трафик для данного хоста (192.168.168.139) в отдельный файл с именем **dump-192.168.168.139.pcap**. Команда для записи выглядит следующим образом:

```
sniffer@sniffer:~ % tshark -i bridge0 -w dump-192.168.168.139.pcap port 80
Capturing on 'bridge0'
```

Запустившись, утилита **tshark** будет сообщать о количестве записанных пакетов в виде постоянно увеличивающегося счетчика. Дождемся, пока пользователь сделает пару HTTP запросов (в рамках данного эксперимента я сам открою окно браузера и зайду на Web-админку устройства). Утилита **tshark** показывает следующее:

```
134 ^C
```

Как только я вошел в «админку», счетчик пакетов резко увеличился. Жмем **Ctrl-C** чтобы остановить запись.

Теперь нам нужно посмотреть какие TCP потоки были записаны и вывести их идентификаторы (числа от нуля и выше), чтобы далее декодировать каждый поток по отдельности. Так как нас интересуют только HTTP запросы и ответы на них, то вводим следующую команду с указанием фильтра:

```
sniffer@sniffer:~ % tshark -n -r dump-192.168.168.139.pcap -2 -R "http.request or
http.response" -T fields -e tcp.stream
0
0
1
3
3
1
4
4
```

Утилита **tshark** выдает нам следующие идентификаторы потоков в которых она обнаружила HTTP трафик: **0**, **1**, **3** и **4**. Запомним их.

Теперь мы можем декодировать трафик каждого потока следующей командой, указав номер потока в самом конце (цифра **0** после «ascii»):

```
sniffer@sniffer:~ % tshark -n -r dump-192.168.168.139.pcap -z follow,tcp,ascii,0
```

Утилита выдает содержимое **0-го** потока:

```
===================================================================
Follow: tcp,ascii
Filter: tcp.stream eq 0
```

```
Node 0: 192.168.176.155:54742
Node 1: 192.168.168.139:80
378
GET / HTTP/1.1
Host: 192.168.168.139
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i


        176
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
Server: SIPHomePhone/2.0
Content-Type: text/html


        168
<html>
<head><meta http-equiv="refresh" content="0; url=admin_new.cgi"></head>
<body><p>The page has moved to: <a href="admin_new.cgi">this page</a></p></body>
</html>

===================================================================
```

Видим, что Web-админка данного устройства, после стандартного запроса «GET /» отфутболила браузер пользователя на другой **url=admin_new.cgi**. Это значит, что браузер сделает еще один запрос. Поэтому декодируем следующий поток, вводим команду для декодирования потока номер **1**:

```
sniffer@sniffer:~ % tshark -n -r dump-192.168.168.139.pcap -z follow,tcp,ascii,1
```

В ответ получаем текст запроса и гигантский «респонз» на него:

```
===================================================================
Follow: tcp,ascii
Filter: tcp.stream eq 1
Node 0: 192.168.176.155:59591
Node 1: 192.168.168.139:80
391
GET /admin_new.cgi HTTP/1.1
Host: 192.168.168.139
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i


        72
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Server: SIPHomePhone/2.0

        1385
Content-type: text/html; charset=utf-8

<html>
<head>
<meta charset="utf-8" />
```

```
<meta http-equiv="cache-control" content="no-cache"/>
<meta http-equiv="pragma" content="no-cache"/>

<link rel='stylesheet' id='theme-link' href='spectre.min.css'>
<link rel='stylesheet' id='iconlib-link' href='spectre-icons.css'>
<style>
#navbar-fabmicro a:hover {background-color:#f7f7f7;}
.menu-active {font-weight:bold;}

…

</script>
</div>
<hr>
<div class="text-center">{SID: <b>1466511685677080494854530CA38107</b>} {Serial: <b>1659112279</b>}
{Source: default_config}</div>
</body>
</html>

======================================================================
```

Из ответа я вырезал большую часть HTML кода страницы, так как в данном случае в нем нет ничего интересного для нас. Всё самое интересно находится в HTTP заголовках, а именно вот в этом:

```
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

Такой заголовок указывает на то, что Web сервер попросил клиента провести авторизацию типа **Basic**, а это означает, что внутри закодированной строки в формате Base64 содержится логин и пароль в открытом тексте. Чтож, осталось только декодировать эту строку следующей командой:

```
sniffer@sniffer:~ % echo YWRtaW46cGFzc3dvcmQ= | base64 -d
admin:password
```

И мы тут же узнаем, что пользователь применил логин: **admin**, и пароль: **password**.

Цель достигнута!

Хочу заметить, что 99% трафика передаваемого по сетям в настоящее время подвергается шифрованию с помощью различных алгоритмов и протоколов (SSL, TLS), поэтому вот так просто подслушать пароли для любого пользовательского потока в современных реалиях просто не возможно. Приведенный выше пример по больше части смоделирован и показан для демонстрации возможностей утилиты **tshark** по извлечению полезной нагрузки из потока трафика.

Но также и замечу, что у этой утилиты есть ключ **-o** с помощью которого можно указать на RSA ключ от SSL сертификата и тогда **tshark** автоматически декодирует и расшифрует SSL/TLS трафик. Таким образом, задача прослушки сводится к тому, где взять ключ. ;-)

6.5. Анализ сетевой нагрузки с помощью утилиты `trafshow’

Еще одна полезная утилита о которой хочется рассказать — это **trafshow**. Эта утилита была написана нашим соотечественником Владимиром Воробьевым из Новосибирского Гос. Университета в далёких 90-х годах прошлого столетия. Утилита по сей день поддерживается и прекрасно работает на многих Unix системах имеющих BPF API. В стандартную поставку ОС FreeBSD она не входит, но находится в репозитории в виде отдельного пакета с одноименным названием, то есть установить её можно одной командой:

```
sniffer@sniffer:~ % sudo pkg install trafshow
```

Подобно **tcpdump** и **tshark**, утилита **trafshow** принимает в качестве параметров командной строки набор BPF правил. Однако задача утилиты не раскладывать трафик до отдельных пакетов, а наоборот — группировать (суммировать трафик) по потокам и отображать текущую интенсивность каждого потока в виде списка. Для каждого потока утилита **trafshow** отображает хост и порт источника и назначения пакетов, а также текущую интенсивность в единицах **CPS** (Characters per Second, то есть «байт в секунду») и сумму накопленного трафика (в байтах) за время наблюдения. Утилита **trafshow** хорошо подходит для мониторинга текущей нагрузки сети, для выявления нежелательных источников трафика, а так же полезна при борьбе с DDoS атаками.

Утилита **trafshow** всегда работает в текстовом интерактивном режиме. Это означает, что пользователь может перемещаться по списку потоков в реальном времени (перемещая курсор вверх или вниз), выбрав интересующий поток и нажав Enter может получать дампы пакетов в HEX и ASCII формате для данного потока. Выход из выбранного потока осуществляется нажатием клавиши Esc. На рис. 10 показано окно терминала в котором запущена утилита **trafshow**.

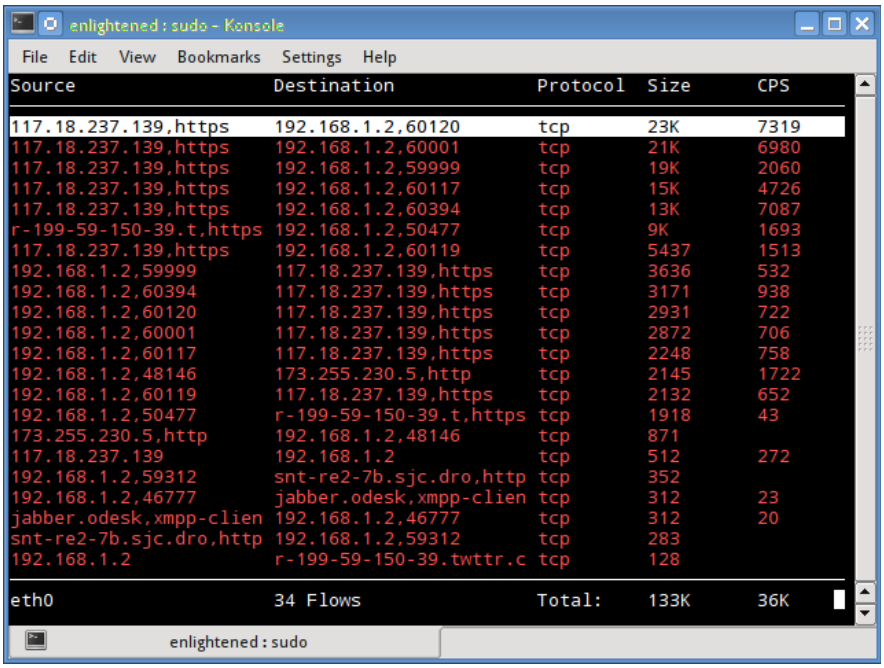

Рис. 10. Окно терминала с запущенной утилитой trafshow. Взято из сети Интернет.

Для того, чтобы продемонстрировать работу **trafshow** я стартую её на снифферe указав ряд правил для исключения лишнего, не интересующего меня, трафика. Лишний

трафик в данном случае это потоки протоколов SSH и GRE (туннель PPTP) через которые я подключаюсь к машине сниффера. Если эти потоки не исключить, то они будут «самоусиливаться», так как поток данных выводимый **trafshow** будет проходить по этому же самому интерфейсу создавая «положительную обратную связь» и в конечном счете забьет весь сетевой интерфейс. Поэтому вводим следующую команду указав фильтр:

```
sniffer@sniffer:~ % trafshow -i bridge0 not port 22 and not arp and not proto gre
```

В другом окне терминала я запущу утилиту **ping** на известный адрес IP-домофона, причем запущу её в режиме «flood» чтобы создать как можно больше ICMP трафика. Для этого введу следующую команду на своем рабочем ПК:

```
rz@butterfly:~ % sudo ping -f 192.168.168.139
Password:
PING 192.168.168.139 (192.168.168.139): 56 data bytes
.....
```

В процессе «Flood Ping» утилита **ping** непрерывно передает в сеть пакеты типа ICMP Echo-Request размером по 56 байт (значение по умолчанию) не дожидаясь ответа на них. Если в течении некоторого времени ответ на переданный пакет не поступает, то такой пакет считается потерянным и утилита **ping** отображает в терминал символ «точка» («.»), что позволяет оценить интенсивность потерь в реальном времени.

Перейдем в окно терминала подключенного к машине сниффера и понаблюдаем на процесс отображаемый в утилите **trafshow**. Мы увидим окно, примерное содержание которого показано на рис. 11.

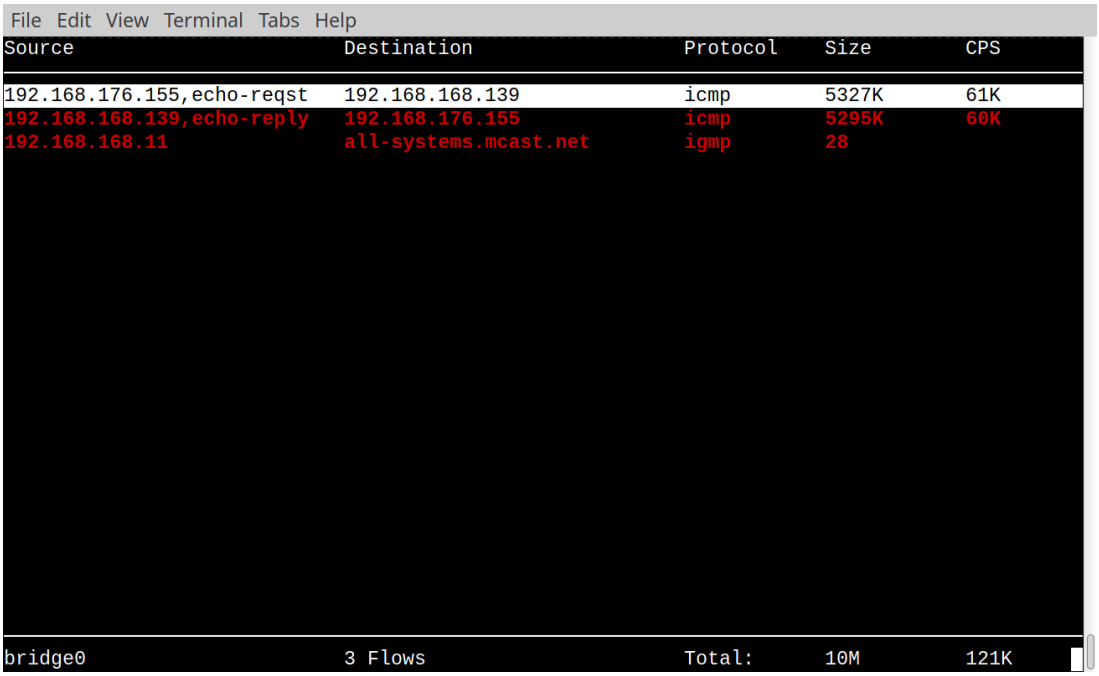

Рис.11. Окно терминала с утилитой trafshow при измерении потока ICMP трафика. «Flood Ping» пакетом по 56 байт.

Мы видим, что **trafshow** зафиксировала три потока: ICMP echo-request, ICMP echo-reply и IGMP. Поток IGMP — это единичный пакет от коммутатора и в нём передано всего 28 байт. А вот потоки ICMP имеют достаточно высокую интенсивность — по 60 КБ/сек (килобайт в

секунду) в каждом направлении. В самой нижней строке утилита **trafshow** показывает общую загрузку сетевого интерфейса и суммарное число переданных байт.

Если сейчас нажать клавишу Enter и войти в первый поток, то утилита **trafshow** начнет отображать непрерывный поток захваченных пакетов. Как это выглядит показано на рис. 12 ниже.

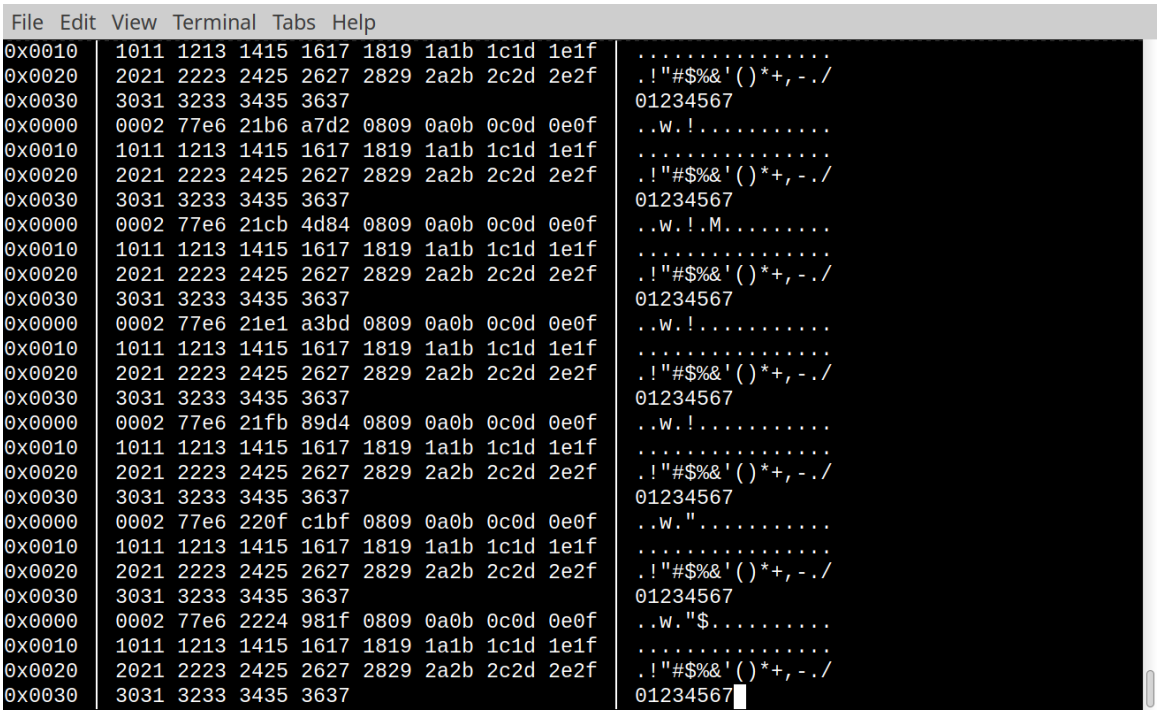

Рис.12. Окно терминала с утилитой trafshow при измерении потока ICMP трафика. Отображение дампа захваченных пакетов.

Остановим «Flood Ping» и запустим его еще раз указав значительно больший размер полезной нагрузки, например 1400 байт:

```
rz@butterfly:~ % sudo ping -s 1400 -f 192.168.168.139
PING 192.168.168.139 (192.168.168.139): 1400 data bytes
.....................
```

Первое что бросается в глаза — интенсивность потерь существенно возросла, это видно по большему числу «точек» и скорости их появления. Перейдем в окно терминала машины сниффера и посмотрим что нам показывает утилита **trafshow**. На рис. 13а видно, что скорость (CPS) существенно выросла и теперь составляет около 460 КБ/сек или 3,6 Мбит/сек, что для сети FastEthernet (именно такой тип Ethernet порта на IP-домофоне) составляет менее 5% её пропускной способности! Можно сделать предположение о том, что либо в сети, либо на устройстве присутствую проблемы.

Мне стало интересно почему так происходит и я выполнил еще один «Flood Ping» тест, в этот раз прямо с машины сниффера (напомню, что IP-домофон соединен со сниффером напрямую UTP патч-кордом). В этом тесте, см. рис. 123, я получил CPS равный 3257 КБ/сек, что составляет около 32% от максимально возможного, при этом за время теста было потеряно всего два пакета (packet loss менее 0,01%). Небольшое исследование показало, что проблема низкой пропускной способности и высокого коэффициента потерь при первых

двух тестах была вызвана сетью WiFi, через который подключен мой рабочий ноутбук. «Век живи - век учись» - говорил герой небезызвестного анекдота. ;)

Рис.13. Окно терминала с утилитой trafshow при измерении потока ICMP трафика: слева (а) при «flood ping» с рабочего ПК через WiFi; справа (б) при «flood ping» напрямую с машины сниффера.

6.6. SNORT® Intrusion Prevention System

Мой рассказ про анализ сетевого трафика был бы неполным без упоминания такой чудесной утилиты, или даже правильнее сказать — системы, как SNORT Intrusion Prevention System. Этот комплекс состоит из утилиты **snort** и большого объема скриптов и плагинов написанных на различных языках, различными людьми из различных компаний.

Рис. 14. Логотип SNORT® Intrusion Prevention System

Разработка SNORT как простой утилиты обнаружения подозрительного трафика была начата в 1998 году небольшим коллективом программистов из компании Sourcefire во главе с Мартином Рёшем (Martin Roesch). В 2009 году утилита SNORT вошла в «Зал Славы СПО» организованный журналом InfoWorld. В 2013 году SNORT и весь коллектив разработчиков был приобретен компанией Cisco Systems, которая по сей день активно развивает этот бесплатный продукт с открытым исходным кодом. Я вряд ли ошибусь, если скажу, что SNORT сейчас является Номер 1 в области детектирования, предупреждения и активной защиты сетей от вторжения. SNORT умело упакован в огромное количество красивых коммерческих железных коробок - «файерволлов» (межсетевых экранов) продаваемых за огромные деньги. При этом сам SNORT остается бесплатным и доступным каждому, кто способен нажимать клавиши на клавиатуре. И да, SNORT входит в репозиторий пакетов ОС FreeBSD, поэтому его легко установить командой:

```
sniffer@sniffer:~ % sudo pkg install snort3
```

Чтобы получить практическую пользу от SNORT, его недостаточно просто установить. Требуется глубокая интеграция с операционной системой (в частности со встроенным в ядро **PF** фильтром), написание своих правил и скриптов. Также потребуется

произвести глубокую перестройку сети, без чего SNORT просто бесполезен. Но всё это доступно инженеру средней руки, главное — уметь читать документацию по SNORT и не бояться экспериментировать. В этой статье я не буду давать какие либо руководства и советы по настройке SNORT, уж сильно это необъятная тема. Моя задача — показать и заинтересовать читателя.

Тем не менее, пару примеров использования SNORT на нашем снифферe я приведу, но прежде немного теории. Утилита **snort** (я буду называть так основную программу этого комплекса) умеет работать в трех режимах:

1. «Сниффер» — анализирует проходящий мимо трафик в попытке отыскать в нем что-то что соответствует заданным правилам, очень похож на (**tshark** с опциями **-Y** и **-R**).
2. «Логгер пакетов» — сохраняет проходящие мимо пакеты в файлы-дампы для последующего анализа. Причем, имеется целая система создания и ведения этих файлов так как записываемого трафика может быть огромное количество и **snort** прекрасно с этим справляется.
3. «Детектор вторжений» (NIDS) - глубоко анализирует проходящий трафик и пытается определить, основываясь на огромной базе постоянно обновляемых правил, попытки вторжения, сканирования портов, спуфинга или DDoS атак. В случае обнаружения (сработки триггеров), **snort** может включать «логгер пакетов», а также автоматически формировать и добавлять правила в **PF** для отсечения подозрительного трафика.

6.6.1. Детектируем сканирование портов одно-строчным правилом для `snort’

Одним из частых предвестников атаки на сетевую инфраструктуру является сканирование TCP и UDP портов в сети (или на машине) жертвы. Атакующий проводит сканирование чтобы выяснить потенциальные слабые места на «теле» жертвы и в последствии атаковать их. Поэтому детектирование процесса сканирования портов и оперативная реакция на выявление такого события является первостепенным делом при грамотно выстроенной защите. Надо заметить, что сам процесс сканирования портов может быть очень изощренным и вообще плохо поддаваться процессу детектирования. Тем не менее мы рассмотрим простой и довольно часто используемый пример — сканирование с помощь отправки пустых TCP пакетов с установленным флагом SYN (запрос на установку соединения). Атакующий генерирует поток таких пакетов при этом перебирая случайным образом номер порта. Хост. на котором есть сервис слушающий на порту, отвечает на запрос в попытке провести хендшейк, так как для него это часть стандартной процедуры, что фиксируется атакующим. Не всякий TCP-SYN пакет с нулевым размером полезной нагрузки является признаком сканирования, но по сложившейся традиции, при установке соединения TCP клиент обычно в первом же пакете высылает кусочек данных будущего запроса. Поэтому появления пустых TCP-SYN является достаточно редким событием. Появление таких пакетов на Вашей сети в большом количестве уже можно считать тревожным сигналом.

С помощью утилиты **snort** очень легко детектировать такие пакет, это можно сделать даже не прибегая к тонким настройкам всей системы SNORT, а просто указав утилите правило прямо в командной строке. Введем следующую команду:

```
sniffer@sniffer:~ % snort -i bridge0 -A fast --rule 'alert tcp any any -> any any (flags:S;
dsize:0; msg:"Possible NMAP TCP port scan"; sid: 1231213;)'
```

Откроем другое окно терминала на рабочем ПК и выполним сканирование TCP портов на IP адресе нашего IP-домофона с помощью утилиты **nmap**:

```
rz@butterfly:~ % sudo nmap -v 192.168.168.139
Starting Nmap 7.94 ( https://nmap.org ) at 2025-07-12 18:12 +05
Initiating Ping Scan at 18:12
Scanning 192.168.168.139 [4 ports]
...
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Видно, что утилита **nmap** обнаружила два открытых порта: **22/tcp** для SSH и **80/tcp** для HTTP (Web-админка). Переключимся в окно машины сниффера и наблюдаем сообщения следующего вида:

```
07/15-02:55:58.000929 [**] [1:1231213:0] "Possible NMAP TCP port scan" [**] [Priority: 0] {TCP}
192.168.176.155:62246 -> 192.168.168.139:4126
07/15-02:55:58.001269 [**] [1:1231213:0] "Possible NMAP TCP port scan" [**] [Priority: 0] {TCP}
192.168.176.155:62246 -> 192.168.168.139:24800
07/15-02:55:58.001488 [**] [1:1231213:0] "Possible NMAP TCP port scan" [**] [Priority: 0] {TCP}
192.168.176.155:62246 -> 192.168.168.139:7106
07/15-02:55:58.001674 [**] [1:1231213:0] "Possible NMAP TCP port scan" [**] [Priority: 0] {TCP}
192.168.176.155:62246 -> 192.168.168.139:2998
07/15-02:55:58.001842 [**] [1:1231213:0] "Possible NMAP TCP port scan" [**] [Priority: 0] {TCP}
192.168.176.155:62246 -> 192.168.168.139:60020
```

Уже неплохо! Но такое одно-строчное правило не может справиться с более изощренными видами сканирования, например с приманкой (decoy). Поэтому в качестве еще одного примера рассмотрим использование встроенного в SNORT 3 плагина с одноименным названием «port_scan».

6.6.2. Детектируем сканирование портов встроенным плагином `port_scan’

Как я уже писал выше, SNORT 3 поставляется с огромны числом различных плагинов среди которых имеется один с названием «port_scan». Это плагин позволяет детектировать далеко не все, но большое количество разновидностей сканирования сети. Чтобы попробовать «port_scan» в действии нам придется немного подредактировать конфигурационный файл утилиты **snort**, так как по умолчанию этот плагин отключен (на то есть определенные причины). Запустим редактор **ee** следующим образом:

```
sniffer@sniffer:~ % sudo ee /usr/local/etc/snort/snort.lua
```

Найдем в файле строку содержащую установку переменной **port_scan =**, раскомментируем её и присвоим значение:

```
port_scan = default_hi_port_scan
```

Далее, ищем строку **ips =** , раскомментруем её и установим следующее значение:

```
ips =
{
    -- use this to enable decoder and inspector alerts
    enable_builtin_rules = true,
}
```

Сохраняем изменения и выходим из **ee**. Запускаем утилиту **snort** со следующими параметрами:

```
sniffer@sniffer:~ % snort -c /usr/local/etc/snort/snort.lua -i bridge0 -A fast
```

После запуска утилита **snort** производит чтение и синтаксическую проверку всех своих конфигурационных файлов (их, кстати, может быть очень много), и подгружает описанные там правила. В конце она выдает некоторую статистику по числу загруженных правил:

```
-------------------------------------------------
pcre counts
               pcre_rules: 1080
              pcre_native: 1080
-------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0    4862       1    4862    /usr/local/etc/snort/snort.lua
-------------------------------------------------
rule counts
       total rules loaded: 4862
          duplicate rules: 1
               text rules: 4236
            builtin rules: 626
            option chains: 4862
            chain headers: 948
                 flowbits: 48
     flowbits not checked: 23
-------------------------------------------------
```

После проверки и загрузки конфигов, утилита **snort** переходит в рабочий режим. Тем временем я открываю новое окно терминала на своем рабочем ПК и с помощью утилиты **nmap** пытаюсь выполнить несколько видов сканирования все того же IP адреса:

```
# TCP-SYNC
rz@butterfly:~ % sudo nmap -v 192.168.168.139

# UDP scan
rz@butterfly:~ % sudo nmap -v -sU 192.168.168.139

# IP scan
rz@butterfly:~ % sudo nmap -v -sO 192.168.168.139
```

Каждый раз при запуске **nmap** в окне сниффера я наблюдаю следующие сообщения соответствующие типу проводимого сканирования:

```
07/12-21:06:02.701981 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}
192.168.168.139:5950 -> 192.168.176.155:61333
07/12-21:06:02.702141 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}
192.168.176.155:61333 -> 192.168.168.139:617
07/12-21:06:02.702208 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}
192.168.168.139:617 -> 192.168.176.155:61333
07/12-21:06:02.702372 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}
192.168.176.155:61333 -> 192.168.168.139:9009
07/12-21:06:02.702434 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}
192.168.168.139:9009 -> 192.168.176.155:61333
07/12-21:06:02.703011 [**] [122:1:1] "(port_scan) TCP portscan" [**] [Priority: 3] {TCP}


07/12-21:04:38.489675 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {UDP}
192.168.176.155:47076 -> 192.168.168.139:34578
07/12-21:04:38.489820 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {UDP}
192.168.176.155:47076 -> 192.168.168.139:17207
07/12-21:04:38.490334 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {ICMP}
192.168.168.139 -> 192.168.176.155
07/12-21:04:38.496755 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {UDP}
192.168.176.155:47076 -> 192.168.168.139:18821
07/12-21:04:39.660300 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {UDP}
192.168.176.155:47078 -> 192.168.168.139:18821
```

```
07/12-21:04:39.660512 [**] [122:17:1] "(port_scan) UDP portscan" [**] [Priority: 3] {UDP}
192.168.176.155:47078 -> 192.168.168.139:17207


07/12-21:03:56.396242 [**] [122:9:1] "(port_scan) IP protocol scan" [**] [Priority: 3] {ICMP}
192.168.168.139 -> 192.168.176.155
07/12-21:03:56.737086 [**] [122:9:1] "(port_scan) IP protocol scan" [**] [Priority: 3] {IP}
192.168.176.155 -> 192.168.168.139
07/12-21:03:57.064270 [**] [122:9:1] "(port_scan) IP protocol scan" [**] [Priority: 3] {IP}
192.168.176.155 -> 192.168.168.139
07/12-21:03:57.385228 [**] [122:9:1] "(port_scan) IP protocol scan" [**] [Priority: 3] {IP}
192.168.176.155 -> 192.168.168.139
```

Как я уже упоминал, утилиту **snort** можно легко настроить на отправку предупреждающих сообщений (alert-ов) по протоколу **syslog** на централизованный сервер реагирования или на запись сообщений в базу данных. Можно настроить автоматическую блокировку трафика от IP адресов вызывающих подозрительный трафик. Но этот материал я оставлю для самостоятельного освоения читателем. :-)

7. Бонус

В этой главе я приведу быстрые инструкции по разворачиванию устройства мониторинга сети (сниффера). Мной был настроен и подготовлен загрузочный образ операционной системы FreeBSD 14.3 и набор скриптов к ней для выполнения всех описанных в данной статье настроечных манипуляций. Также написан простенький «cheat-sheet» и ряд других инструкций в формате Markdown. Всё это собрано в одном репозитории и выложено на Github для Вашего удобства. Поэтому, выполняем команду:

```
git clone https://github.com/pointcheck/sniffer.git
```

и читаем README.md.

Прямая ссылка на загрузочный образ:

```
https://github.com/pointcheck/sniffer/raw/refs/heads/master/Sniffer-FreeBSD-14.3-bootable.img.xz
```

Пожатый образ занимает около **1ГБ** дискового пространства.

7.1. Инструкция по быстрой установке сниффера на базе ОС FreeBSD

1. Вставьте свободный USB Flash накопитель в USB порт Вашего рабочего ПК. Потребуется накопитель размером не более 5ГБ.

2. Распакуйте скачанный образ и запишите его на USB носитель следующей командой:

   Для ОС FreeBSD:

   ```
   sudo sh -c "xz -d -c Sniffer-FreeBSD-14.3-bootable.img.xz | dd of=/dev/da0 bs=1m conv=sync status=progress"
   ```

   Для ОС Linux :

   ```
   sudo sh -c "xz -d -c Sniffer-FreeBSD-14.3-bootable.img.xz | dd of=/dev/sdb bs=1m conv=sync status=progress"
   ```

   Разница только в именовании устройства USB накопителя.

   Пользователям Windows придется заморочиться с **Rufus** или аналогичной утилитой.

3. После окончания записи переставьте USB накопитель из своего ПК в мини-ПК, из которого Вы собираетесь сделать сниффер, и выполните загрузку мини-ПК с этого USB носителя. Снифром может выступать любая x86 совместимая машина с двумя Ethernet портами, один из портов может быть внешним USB-Ethernet. Не забудьте зайти в BIOS и установить приоритет загрузки — с USB носителя.

4. После загрузки ОС FreeBSD зайдите в систему под пользователем **sniffer** используя пароль **reffins**. На всякий случай пароль root-а: **1234**. Поменяйте его на более правильный, с Вашей точки зрения, командой **passwd**.

5. Запустите скрипт для настройки WiFi чтобы иметь удаленный доступ к машине сниффера:

```
sudo ./sniffer_setup_wifi.pl
```

Скрипт просканирует радиоэфир и покажет доступные SSID идентификаторы WiFi точек, после чего попросит ввести имя одной из них, следом попросит ввести **psk** пароль. Скрип сохранит эти данные в **/etc/wpa_supplicant.conf** и перезапустит сеть.

6. Подключите оба Ethernet порта мини-ПК в разрез между исследуемым устройством (DUT) и сетью используя два UTP патч-корда. Запустите скрипт для настройки Ethernet Bridging-а:

```
sudo ./sniffer_setup_bridge.pl
```

Скрип отобразит имеющиеся в Вашей системе сетевые интерфейсы и попросит ввести имена сетевых интерфейсов (по одному за раз), которые требуется объединить в сетевой мост. В результате создаст новый сетевой интерфейс **bridge0**, добавит соответствующие строки в конфигурационный файл **/etc/rc.conf** и перезапустит сетевой стек и сервисы. В этот момент, при подключении через WiFi, у Вас может пропасть соединение с машиной.

7. Запустите скрипт (опционально) для настройки **mpd5** на организацию PPTP, L2TP и PPPOE туннелей для удаленного подключения к машине сниффера из Интернет:

```
sudo ./sniffer_setup_mpd5.pl
```

Скрипт попросит Вас выбрать тип туннеля (можно выбрать сразу все три), указать имя пользователя, пароль и IP адрес сервера. В результате скрипт создаст конфигурационный файл **/usr/local/etc/mpd5/mpd.conf** и положит рядом два скрипта **ifup.sh** и **ifdown.sh**.

Важно! Данный скрипт настроит систему на использование **нескольких таблиц маршрутизации (FIB)**, это стандартный способ в ОС FreeBSD организации «source routing». Необходимо это для того, чтобы все три туннеля не мешали друг другу, так как каждый из них будет иметь свой «default route». Скрипт **ifup.sh** будет запускать копию **sshd** для обслуживания соединений только заданного туннеля. Если у Вас настроены и подключены все три типа туннелей, то в системе будет четыре копии **sshd** (одна копия системная, на нулевом FIB).

8. Перезагрузите машину сниффера командой:

```
sudo reboot
```

9. Еще раз войдите в систему (подключитесь через **ssh** или с консоли). В Вашем распоряжении утилиты **tcpdump**, **tshark**, **trafshow** и **snort**.

10. Короткий «cheat-sheet» находится в файле **cheasheet.md**. Просмотреть содержимое файла можно командой:

```
more cheatsheet.md
```

11. Рекомендую подвесить «cheat-sheet» в качество «Message of The Day» чтобы он отображался при каждом входе в систему. Сделать это можно следующей последовательностью команд:

```
sudo cp cheatsheet.md /etc/motd.template
sudo service motd restart
```

12. Несколько инструкций с типовыми операциями находятся в **.md** файлах в каталоге у пользователя **sniffer**, прочитать их можно следующими командами:

```
more rule_sip.md
more rule_ssh.md
more rule_http_password.md
more snort.md
```

7.2. Инструкция по созданию своего загрузочного образа ОС FreeBSD со сниффером

После некоторого времени работы со сниффером у Вас ожидаемо возникнет желание сделать свой (на базе текущей системы) загрузочный образ для USB накопителя. Сделать это помогут следующие два скрипта:

1. Чтобы создать загрузочный USB накопитель на базе текущей системы, подключитесь к машине сниффера, при этом операционная система должна быть загружена с SSD или HDD (**/dev/ada0**). Вставьте свободный USB накопитель и запустите следующий скрипт:

   ```
   sudo ./make_bootable_usb.sh
   ```

   Этот скрип полностью уничтожит данные на USB, пересоздаст новую таблицу разделов, выполнит разметку файловых систем, добавить EFI загрузчик и скопирует текущую систему. После выполнения этого скрипта у Вас появится еще один загружаемый USB накопитель.

2. Чтобы сделать файл-образ (**.img.xz**) с этого USB накопителя, который Вы сможете выложить на Github или поделиться им с друзьями, выполните следующий скрипт:

   ```
   sudo ./make_image.sh
   ```

На этом, пожалуй, всё!

С уважением, Руслан.