Отчёт о запуске eBPF-топологии в среде Mininet

3. Ход выполнения

3.1. Настройка Mininet

Mininet был установлен в виртуальной машине Linux. После установки пакетов **clang**, **llvm**, **make**, **libelf-dev**, **linux-headers** и **bpftool** была подготовлена среда для компиляции и загрузки eBPF-программ. Для верного подбора заголовков указывается версия ядра системы:

```
sudo apt update
sudo apt install -y clang llvm libelf-dev libbpf-dev linux-headers-$(uname -
r) make bpftool
```

Основной задачей топологии на Kathara было смоделировать DDoS-атаку. Хост, принимающий трафик, загружает eBPF-программу, которая должна отразить угрозу в UDP-потоке от второго хоста.
Соответственно, в сети Mininet были также созданы приемник и источник трафика:

- **h1** — приёмник, на котором должна загружаться XDP/eBPF-программа;
- **h2** — источник UDP-потока (имитация DDoS-нагрузки).

```
#!/usr/bin/env python3
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, OVSController
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import setLogLevel, info
import os

class Router(Node):
    "Маршрутизатор с включённым IPv4 форвардингом"
    def config(self, **params):
        super(Router, self).config(**params)
        self.cmd('sysctl -w net.ipv4.ip_forward=1')
    def terminate(self):
        self.cmd('sysctl -w net.ipv4.ip_forward=0')
        super(Router, self).terminate()
```

```
class XdpTopo(Topo):
    def build(self):
        # Хосты
        h1 = self.addHost('h1')  # 195.11.14.5/24
        h2 = self.addHost('h2')  # 200.1.1.7/24
        # Роутеры
        r1 = self.addNode('r1', cls=Router)
        r2 = self.addNode('r2', cls=Router)
        # Линки: h1--r1--r2--h2
        self.addLink(h1, r1, cls=TCLink)
        self.addLink(r1, r2, cls=TCLink)
        self.addLink(r2, h2, cls=TCLink)

def post_configure(net, attach_xdp=True):
    "Назначаем IP/маршруты после net.start(), как ты просила"

    h1, h2 = net.get('h1', 'h2')
    r1, r2 = net.get('r1', 'r2')

    # Имена интерфейсов (Mininet их создаёт последовательно)
    h1_if = h1.intfList()[0].name      # h1-eth0
    r1_if_h1 = r1.intfList()[0].name   # r1-eth0 (к h1)
    r1_if_r2 = r1.intfList()[1].name   # r1-eth1 (к r2)
    r2_if_r1 = r2.intfList()[0].name   # r2-eth0 (к r1)  (порядок может
отличаться!)
    r2_if_h2 = r2.intfList()[1].name   # r2-eth1 (к h2)
    h2_if = h2.intfList()[0].name      # h2-eth0

    # Чтобы не гадать порядок на r2 — привяжем адреса по месту назначения:
    # Удобнее определить по peer name, но проще — посмотреть ip link в CLI,
если понадобится.

    info('*** Назначаем IP-адреса\n')
    # h1 /24
    h1.cmd(f'ifconfig {h1_if} 195.11.14.5/24 up')
    # r1: eth0 к h1, eth1 к r2
    r1.cmd(f'ifconfig {r1_if_h1} 195.11.14.1/24 up')
    r1.cmd(f'ifconfig {r1_if_r2} 100.0.0.9/30 up')
    # r2: eth0 к r1, eth1 к h2 (поменяй, если порядок иной)
    r2.cmd(f'ifconfig {r2_if_r1} 100.0.0.10/30 up')
    r2.cmd(f'ifconfig {r2_if_h2} 200.1.1.1/24 up')
    # h2 /24
    h2.cmd(f'ifconfig {h2_if} 200.1.1.7/24 up')

    info('*** Прописываем шлюзы/маршруты\n')
```

```
    h1.cmd('route add default gw 195.11.14.1')
    h2.cmd('route add default gw 200.1.1.1')
    r1.cmd('ip route add 200.1.1.0/24 via 100.0.0.10 dev ' + r1_if_r2)
    r2.cmd('ip route add 195.11.14.0/24 via 100.0.0.9 dev ' + r2_if_r1)

    # (опционально) ARP статикой, если хочешь устойчивости на pps:
    # h1.cmd(f'arp -s 195.11.14.1 {r1.MAC(r1_if_h1)}')
    # h2.cmd(f'arp -s 200.1.1.1 {r2.MAC(r2_if_h2)}')

    if attach_xdp:
        info('*** Пытаемся повесить XDP на h1\n')
        # предполагается, что в каталоге запуска есть xdp_tree.o и/или
Makefile
        if not os.path.exists('xdp_tree.o') and os.path.exists('Makefile'):
            info('*** Компилируем XDP (make)\n')
            os.system('make')
        if os.path.exists('xdp_tree.o'):
            # generic-режим — работает везде
            out = h1.cmd(f'ip link set dev {h1_if} xdpgeneric obj xdp_tree.o
sec xdp 2>&1')
            info(out)
            info(h1.cmd('bpftool prog show | grep count_udp_flows || true'))
            info(h1.cmd('bpftool map show || true'))
        else:
            info('*** xdp_tree.o не найден — пропускаем attach\n')

def run():
    setLogLevel('info')
    topo = XdpTopo()
    net = Mininet(topo=topo, controller=None, link=TCLink, autoSetMacs=True,
autoStaticArp=False)
    net.start()
    try:
        post_configure(net, attach_xdp=True)
        info('*** Тест пинга h1<->h2\n')
        info(net.ping([net['h1'], net['h2']], 1))
        info('*** Готово. Откройте CLI для ручных тестов (udp_sender,
bpftool, tcpdump)\n')
        CLI(net)
    finally:
        net.stop()

if __name__ == '__main__':
    run()
```

Между узлами должен успешно осуществлялся обмен пакетами. В качестве проверки используем `ping` и `tcpdump`.
Скрипт атаки аналогичен симуляции в Kathara. UDP-трафик генерировался командой:

```
h2 python3 udp_sender.py
```

На принимающем узле `h1` с помощью:

```
tcpdump -i h1-eth0 udp port 9999
```

Команда фиксирует приходящие пакеты, из чего формируется вывод о корректности маршрутизации и работоспособности сети.

3.2. Сборка eBPF-программы

В каталоге `/home/mininet/lab_ebpf` узла-приёмника находился исходный файл `xdp_tree.c` и `Makefile`. Скрипты также совпадают с симуляцией в Kathara.
При попытке компиляции командой `make` возникла ошибка:

```
fatal error: 'asm/types.h' file not found
```

Проблема заключалась в том, что Makefile был ориентирован на архитектуру **x86-64** и содержал жёсткий путь:

```
-I/usr/include/x86_64-linux-gnu
```

После установки недостающих заголовков (`linux-libc-dev`, `linux-headers`) компиляция прошла корректно, и был получен объектный файл `xdp_tree.o`.

3.3. Попытка загрузки XDP-программы

После запуска скрипта выполнялась загрузка eBPF-программы в сетевой интерфейс узла-приёмника:

```
sudo ip link set dev h1-eth0 xdp obj xdp_tree.o sec xdp
```

Команда завершилась ошибкой:

```
Error fetching program/map: Operation not supported
```

Попытка использовать **generic-режим**:

```
sudo ip link set dev h1-eth0 xdpgeneric obj xdp_tree.o sec xdp
```

завершилась без фатальной ошибки, однако программа не появилась в списке активных (`bpftool prog show` возвращал пустой результат).
Это свидетельствует о том, что eBPF-код не был успешно загружен в ядро.

Ручное монтирование ресурсов не решило проблему с загрузкой кода в

```
sudo mkdir -p /sys/fs/bpf
sudo mount -t bpf bpffs /sys/fs/bpf || true
mount | grep /sys/fs/bpf
```

Таким образом, попытка загрузки eBPF-программы в Mininet оказалась неуспешной из-за возможных архитектурных ограничений среды.
Однако в ходе работы удалось изучить механизм XDP, процесс компиляции и типичные причины ошибок при работе с виртуальными интерфейсами.