Progetto #3 del corso di Network and System Defense

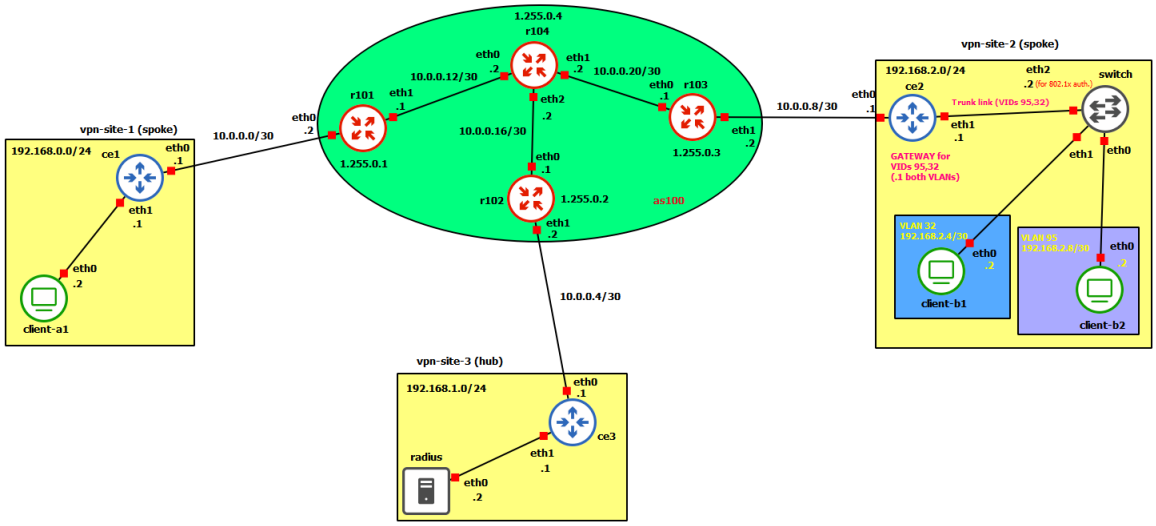
Stefano Belli, matricola 0350116

Anno accademico 2024/2025

Indice

Introduzione

Il progetto prevede l'implementazione di una VPN MPLS/BGP hub-and-spoke con possibilità di comunicazione spoke-to-spoke e advertisement delle rotte CE-PE eseguita automaticamente. In aggiunta, nella VPN site 1 c'è un dispositivo sensitive, quindi bisogna impiegare un mandatory access control, nella VPN site 2 ci sono due client che possono accedere alla LAN solo dopo autenticazione (802.1x), e quindi ricevere un VLAN id automaticamente. Nella VPN site 3 c'è il server RADIUS che è l'authentication server.



Configurazione della backbone (AS100)

Nota: per le conf. dei nodi è indicato il nome del file, prima del suo contenuto.

Il nome è relativo a /root per quanto riguarda i container, oppure alla directory del dump degli script: project/scripts/nome-sito/nome-device

I nomi dei dispositivi e dei sites sono in lowercase sia nella topologia che nella directory del dump degli script.

- **R101**

Configurazione FRR

Configurazione di IP sulle interfacce che connettono il provider edge agli altri router (CE-1 e LSR).

Interfaccia di loopback utilizzata per resilienza ulteriore a guasti.

OSPF come IGP funzionante nell'interfaccia che lo collega all'LSR.

MPLS viene utilizzato come data plane della VPN, double encapsulation all'invio verso un'altro site, in ricezione label più interna identifica VRF da utilizzare per la consegna (non è il caso, ma che succede se in cui il provider fornisce VPN a più customer e questi usano stessi IP per le subnet?).

MP-iBGP configurato tra i provider edge. Si tratta del control plane della VPN.

Le route distinguisher servono a distinguere le VPN (come prima, non è questo il caso ma potrebbero essere serviti molteplici customer).

Ancora più importante è il route target: per implementare la hub-and-spoke topology - manteniamo tutte le connessioni tra i provider edge (MP-iBGP), ma filtriamo le rotte da imparare.

Questo provider edge esporta le rotte con route-target 100:1 e importa solo rt 100:2 (PE hub), è uno spoke.

La stessa cosa fa l'altro provider edge spoke: tutti e due importeranno solamente le rotte diffuse dal PE hub.

`frrconf`

```
interface eth0
 ip address 10.0.0.2/30

interface eth1
 ip address 10.0.0.13/30

interface lo
 ip address 1.255.0.1/32

router ospf
 router-id 1.255.0.1
 network 1.255.0.1/32 area 0
 network 10.0.0.12/30 area 0
 exit

mpls ldp
 router-id 1.255.0.1
 ordered-control
 address-family ipv4
  discovery transport-address 1.255.0.1
  interface lo
  interface eth1

router bgp 100
 bgp router-id 1.255.0.1
 neighbor 1.255.0.2 remote-as 100
```

```
  neighbor 1.255.0.2 update-source 1.255.0.1
  neighbor 1.255.0.3 remote-as 100
  neighbor 1.255.0.3 update-source 1.255.0.1
  address-family ipv4 unicast
   neighbor 1.255.0.2 next-hop-self
   neighbor 1.255.0.3 next-hop-self
  address-family ipv4 vpn
   neighbor 1.255.0.2 activate
   neighbor 1.255.0.2 next-hop-self
   neighbor 1.255.0.3 activate
   neighbor 1.255.0.3 next-hop-self

router bgp 100 vrf vpnA
 neighbor 10.0.0.1 remote-as 65000
 address-family ipv4 unicast
  label vpn export auto
  rd vpn export 100:0
  rt vpn export 100:1
  rt vpn import 100:2
  export vpn
  import vpn
```

Script di init

Eseguirlo per configurare la VRF associata alla vpnA nel kernel, abilitare funzionalità MPLS e configurare FRR.

`net.sh`

```
#!/bin/bash

ip link add vpnA type vrf table 10
ip link set vpnA up
ip link set eth0 master vpnA

sysctl -w net.mpls.conf.lo.input=1
sysctl -w net.mpls.conf.eth1.input=1
sysctl -w net.mpls.conf.vpnA.input=1
sysctl -w net.mpls.platform_labels=100000

vtysh -f frrconf
```

- **R102**

Configurazione di FRR

Configurazione del PE associato all'hub della VPN.

Del tutto simile alla precedente, se non fosse che bisogna permettere la spoke-to-spoke communication attraverso l'hub.

Le route target sono import 100:1 e export 100:2, in altre parole importiamo rotte da entrambi gli spoke e esportiamo le nostre verso gli spoke (per come è configurata la rete).

Redistribute kernel è necessario perchè la route di default è configurata con il tool "ip" invece che con FRR, che la vede come "K", ovvero, kernel. Questo permette l'esportazione di 0.0.0.0/0 verso gli spoke e quindi di abilitare la spoke-to-spoke comm "through the hub" (0.0.0.0/0 via CE dell'hub).

In particolare, avendo esportato la rotta 0.0.0.0/0 agli altri spoke, quando uno vuole parlare con l'altro, nelle rispettive VRF, osserverà che l'unico match dell'IP.dest nelle routing tables è con 0.0.0.0/0 che l'hub ha esportato (gli spoke conoscono solo le rotte esportate dall'hub e quella del "CE associato" tramite route advertisement automatice CE-PE) quindi inoltrano il pacchetto con IP.dest = "dispositivo in altra VPN site spoke" al PE hub che poi lo inoltra al CE dell'hub che lo rimanda al PE hub.

Il PE hub conosce tutte le rotte e quindi sa come inoltrarlo allo spoke di destinazione.

`frrconf`

```
interface eth1
 ip address 10.0.0.6/30

interface eth0
 ip address 10.0.0.17/30

interface lo
 ip address 1.255.0.2/32

router ospf
 router-id 1.255.0.2
 network 1.255.0.2/32 area 0
 network 10.0.0.16/30 area 0
 exit

mpls ldp
 router-id 1.255.0.2
 ordered-control
 address-family ipv4
  discovery transport-address 1.255.0.2
  interface lo
  interface eth0

router bgp 100
 bgp router-id 1.255.0.2
 neighbor 1.255.0.1 remote-as 100
 neighbor 1.255.0.1 update-source 1.255.0.2
 neighbor 1.255.0.3 remote-as 100
 neighbor 1.255.0.3 update-source 1.255.0.2
 address-family ipv4 unicast
  neighbor 1.255.0.1 next-hop-self
  neighbor 1.255.0.3 next-hop-self
 address-family ipv4 vpn
  neighbor 1.255.0.1 activate
```

```
  neighbor 1.255.0.1 next-hop-self
  neighbor 1.255.0.3 activate
  neighbor 1.255.0.3 next-hop-self

router bgp 100 vrf vpnA
 neighbor 10.0.0.5 remote-as 65001
 address-family ipv4 unicast
  redistribute kernel
  label vpn export auto
  rd vpn export 100:0
  rt vpn import 100:1
  rt vpn export 100:2
  import vpn
  export vpn
```

Script di init

Eseguirlo per configurare la VRF associata alla vpnA nel kernel, abilitare funzionalità MPLS, configurare FRR e installare default route verso il CE dell'hub per la VRF associata alla vpnA.

`net.sh`

```
#!/bin/bash

ip link add vpnA type vrf table 10
ip link set vpnA up
ip link set eth1 master vpnA

sysctl -w net.mpls.conf.lo.input=1
sysctl -w net.mpls.conf.eth0.input=1
sysctl -w net.mpls.conf.vpnA.input=1
sysctl -w net.mpls.platform_labels=100000

vtysh -f frrconf

# default route, enables spoke-to-spoke comm. through hub
ip route add 0.0.0.0/0 via 10.0.0.5 vrf vpnA
```

- **R103**

Configurazione di FRR

Si tratta di un PE spoke.

Configurazione identica a quella dell'altro spoke.

Route target importa rotte 100:2 (PE hub) e esporta 100:1 (PE spoke).

Finalizzando quindi la configurazione hub-and-spoke.

frrconf

```
interface eth1
 ip address 10.0.0.10/30

interface eth0
 ip address 10.0.0.21/30

interface lo
 ip address 1.255.0.3/32

router ospf
 router-id 1.255.0.3
 network 1.255.0.3/32 area 0
 network 10.0.0.20/30 area 0
 exit

mpls ldp
 router-id 1.255.0.3
 ordered-control
 address-family ipv4
  discovery transport-address 1.255.0.3
  interface lo
  interface eth0

router bgp 100
 bgp router-id 1.255.0.3
 neighbor 1.255.0.1 remote-as 100
 neighbor 1.255.0.1 update-source 1.255.0.3
 neighbor 1.255.0.2 remote-as 100
 neighbor 1.255.0.2 update-source 1.255.0.3
 address-family ipv4 unicast
  neighbor 1.255.0.1 next-hop-self
  neighbor 1.255.0.2 next-hop-self
 address-family ipv4 vpn
  neighbor 1.255.0.1 activate
  neighbor 1.255.0.1 next-hop-self
  neighbor 1.255.0.2 activate
  neighbor 1.255.0.2 next-hop-self

router bgp 100 vrf vpnA
 neighbor 10.0.0.9 remote-as 65002
 address-family ipv4 unicast
  label vpn export auto
  rd vpn export 100:0
  rt vpn export 100:1
  rt vpn import 100:2
  import vpn
  export vpn
```

Script di init

Eseguirlo per configurare la VRF associata alla vpnA nel kernel, abilitare funzionalità MPLS e configurare FRR.

net.sh

```
#!/bin/bash

ip link add vpnA type vrf table 10
ip link set vpnA up
ip link set eth1 master vpnA

sysctl -w net.mpls.conf.lo.input=1
sysctl -w net.mpls.conf.eth0.input=1
sysctl -w net.mpls.conf.vpnA.input=1
sysctl -w net.mpls.platform_labels=100000

vtysh -f frrconf
```

- **R104**

Configurazione di FRR

Configurazione del label switched router della rete.

Scambia le label più esterne dei frame MPLS e ne effettua l'inoltro verso il prossimo router di competenza.

frrconf

```
interface eth0
 ip address 10.0.0.14/30

interface eth2
 ip address 10.0.0.18/30

interface eth1
 ip address 10.0.0.22/30

interface lo
 ip address 1.255.0.4/32

router ospf
 router-id 1.255.0.4
 network 1.255.0.4/32 area 0
 network 10.0.0.20/30 area 0
 network 10.0.0.16/30 area 0
 network 10.0.0.12/30 area 0
 exit

mpls ldp
 router-id 1.255.0.4
 ordered-control
```

```
 address-family ipv4
  discovery transport-address 1.255.0.4
  interface lo
  interface eth0
  interface eth1
  interface eth2
```

Script di init

Eseguirlo per configurare funzionalità MPLS nel kernel e FRR.

`net.sh`

```
#!/bin/bash

sysctl -w net.mpls.conf.lo.input=1
sysctl -w net.mpls.conf.eth0.input=1
sysctl -w net.mpls.conf.eth1.input=1
sysctl -w net.mpls.conf.eth2.input=1
sysctl -w net.mpls.platform_labels=100000

vtysh -f frrconf
```

Configurazione della VPN site 1

- **CE1**

Configurazione FRR

Viene configurato, in FRR, l'IPv4 su entrambe le interfacce e sopratutto il route advertisement automatico verso il provider edge (con BGP) della subnet della VPN site 1 (192.168.0.0/24).

`frrconf`

```
interface eth0
 ip address 10.0.0.1/30

interface eth1
 ip address 192.168.0.1/24

ip route 0.0.0.0/0 10.0.0.2

router bgp 65000
 network 192.168.0.0/24
 neighbor 10.0.0.2 remote-as 100
```

Script di init

Eseguirlo per iniziallizare FRR

`net.sh`

```
#!/bin/bash

vtysh -f frrconf
```

- **client-A1**

Script di init

Eseguirlo per configurare IPv4

`net.sh`

```
#!/bin/bash

ip addr add 192.168.0.2/24 dev eth0
ip route add default via 192.168.0.1 dev eth0
```

Configurazione AppArmor come MAC

Sfruttiamo i profili di AppArmor per il MAC. Sono stati realizzati 6 profili.

- *usr.bin.cat*: che impedisce al programma cat di leggere file di autenticazione del sistema.
- *usr.bin.curl*: che impedisce a curl di scrivere file su qualsiasi directory che non sia /tmp (file eliminati al reboot) evitando di riempire il sistema con file inutili e mai più eliminati o utilizzati.
- *usr.bin.rm*: che impedisce all'utente di fare danni al sistema (ad esempio se viene eseguito uno script malevolo con UID=0) impedendone l'accesso alle directory più importanti.
- *usr.bin.wireshark*, *usr.bin.zenmap*, *usr.bin.nmap*: le applicazioni non possono proprio essere caricate (default policy di AppArmor: deny e le regole sono vuote - deny all) vogliamo evitare scanning di qualsiasi tipo sulla rete.

Script di applicazione dei profili AppArmor

Eseguirlo per permettere ad AppArmor di memorizzare i profili nel kernel e impostare la enforcement mode (le policy vengono effettivamente fatte rispettare, al contrario della complain mode, dove vengono solo loggate le violazioni). Applica tutti i profili che trova nella sottodirectory "aaprofs"

`mac/apply-aaprofs.sh`

```
#!/bin/bash
```

```
PROFSDIR=aaprofs
PROFILES=$(ls $PROFSDIR)

echo "Entering directory $PROFSDIR..."
echo "================="
echo "-----------------"

cd $PROFSDIR

for profile in $PROFILES; do
        echo "Applying AppArmor profile: $profile"
        cp $profile /etc/apparmor.d/$profile
        apparmor_parser /etc/apparmor.d/$profile
        apparmor_parser -r /etc/apparmor.d/$profile
        aa-enforce /etc/apparmor.d/$profile
        echo "-----------------"
done

echo "================="
echo "Exiting directory $PROFSDIR..."

cd ..
```

mac/aaprofs/usr.bin.cat

```
#include <tunables/global>

/usr/bin/cat {
    #include <abstractions/base>

    /** r,

    deny /etc/passwd rwxlk,
    deny /etc/group rwxlk,
    deny /etc/shadow rwxlk,
    deny /etc/gshadow rwxlk,
    deny /etc/login.defs rwxlk,
    deny /var/log/lastlog rwxlk,

    /usr/bin/cat mr,
}
```

mac/aaprofs/usr.bin.curl

```
#include <tunables/global>

/usr/bin/curl {
    #include <abstractions/base>
```

```
    #include <abstractions/nameservice>
    #include <abstractions/openssl>

    /usr/bin/curl mr,

    network inet stream,
    network inet6 stream,

    /tmp/* rw,
}
```

mac/aaprofs/usr.bin.nmap

```
/usr/bin/nmap {

}
```

mac/aaprofs/usr.bin.rm

```
#include <tunables/global>

/usr/bin/rm {
    #include <abstractions/base>

    /** rw,

    deny /boot/** w,
    deny /bin/** w,
    deny /usr/bin/** w,
    deny /usr/lib/** w,
    deny /usr/lib64/** w,
    deny /usr/include/** w,
    deny /usr/libexec/** w,
    deny /usr/share/** w,
    deny /usr/sbin/** w,
    deny /lib/** w,
    deny /lib64/** w,
    deny /etc/** w,
    deny /var/** w,
    deny /sbin/** w,

    /usr/bin/rm mr,
}
```

mac/aaprofs/usr.bin.wireshark

```
/usr/bin/wireshark {

}
```

`mac/aaprofs/usr.bin.zenmap`

```
/usr/bin/zenmap {

}
```

Configurazione della VPN site 2

- **CE2**

Configurazione FRR

Viene configurato, in FRR, l'IPv4 su entrambe le interfacce e sopratutto il route advertisement automatico verso il provider edge (con BGP) della subnet della VPN site 2 (192.168.2.0/24).

`frrconf`

```
interface eth0
 ip address 10.0.0.9/30

interface eth1
 ip address 192.168.2.1/24

ip route 0.0.0.0/0 10.0.0.10

router bgp 65002
 network 192.168.2.0/24
 neighbor 10.0.0.10 remote-as 100
```

Script di init

Eseguirlo per iniziallizare FRR e configurare il router per le due VLAN.

Le due VLAN sono VID=95 e VID=32.

Le subnet associate ai due broadcast domain sono:

- 192.168.2.8/30
- 192.168.2.4/30

La scelta di utilizzare due subnet con mask /30 è motivata:

- Non c'è bisogno di andare a modificare ulteriormente il CE per l'advertisement di rotte ulteriori
- Non si intasa il "namespace" della VPN (ogni site ha subnet 192.168.0.0/24, 192.168.1.0/24, ...)
- Di fatto i due dispositivi sono in VPN site 2, ha senso poterli "identificare" con IPv4 coerenti rispetto al "blocco" di indirizzi IP assegnato alla VPN site (e.g. se il blocco è 192.168.2.0/24, identificarlo come 192.168.2.6 è meglio, invece che 192.168.3.1 o 10.0.0.1)
- Longest prefix match - il router sa perfettamente identificare a quale subnet inoltrare

Di contro, "togliamo" gli indirizzi delle subnet /30 alla subnet "principale" /24.

Il resto è configurazione del trunk link, e assegnazione IP alle interfacce che supportano VLAN (router per queste ultime).

`net.sh`

```
#!/bin/bash

vtysh -f frrconf

sysctl -w net.ipv4.ip_forward=1

ip link add link eth1 name eth1.95 type vlan id 95
ip link add link eth1 name eth1.32 type vlan id 32
ip link set eth1.95 up
ip link set eth1.32 up

VLAN_95_IPADDR=192.168.2.9/30
VLAN_32_IPADDR=192.168.2.5/30

#VLAN_95_IPADDR=192.168.4.1/24
#VLAN_32_IPADDR=192.168.3.1/24

ip addr add $VLAN_95_IPADDR dev eth1.95
ip addr add $VLAN_32_IPADDR dev eth1.32
```

- **RADIUS**

Script di init

Eseguirlo per configurare IPv4 e RADIUS.

Il server di autenticazione è RADIUS ed è la classica config basata su MD5.

Unica problematica riscontrata: il file "users" non veniva letto da freeradius, è stato utilizzato il file "authorize" che invece viene correttamente aperto da freeradius, parsato e permette di convalidare l'auth request.

Lo script copia comunque il file authorize sia come */etc/freeradius/3.0/users* che come */etc/freeradius/3.0/mods-config/files/authorize*.

net.sh

```
#!/bin/bash

ip addr add 192.168.1.2/24 dev eth0
ip route add default via 192.168.1.1 dev eth0

INSTALLDIR=/etc/freeradius/3.0
install -D -m444 clients.conf $INSTALLDIR/clients.conf

# apparently on newer freeradius versions
# the "users" file is actually located at
# /etc/freeradius/3.0/mods-config/files/authorize

# comment one of the following lines if one config file
# keeps disturbing freeradius, but should ensure compat
# across freeradius versions / configs

install -D -m444 authorize $INSTALLDIR/mods-config/files/authorize
install -m444 authorize $INSTALLDIR/users
```

radius.sh

```
#!/bin/bash

freeradius $@
```

Identificazione del client RADIUS (switch)

clients.conf

```
client vs2switch {
 ipaddr = 192.168.2.2
 secret = "mysecretpasswd"
 shortname = authnserv
}
```

Identificazione dei client finali da autenticare (supplicants)

Di particolare importanza è il Tunnel-Private-Group-ID che è la VLAN id da assegnare.

authorize (aka users)

```
clientb1 Cleartext-Password := "clientb1passwd"
        Service-Type = Framed-User,
```

```
        Tunnel-Type = 13,
        Tunnel-Medium-Type = 6,
        Tunnel-Private-Group-ID = 32

clientb2 Cleartext-Password := "clientb2passwd"
        Service-Type = Framed-User,
        Tunnel-Type = 13,
        Tunnel-Medium-Type = 6,
        Tunnel-Private-Group-ID = 95
```

Configurazione della VPN site 3

- **CE3**

Configurazione FRR

Viene configurato, in FRR, l'IPv4 su entrambe le interfacce e sopratutto il route advertisement automatico verso il provider edge (con BGP) della subnet della VPN site 3 (192.168.1.0/24).

`frrconf`

```
interface eth0
 ip address 10.0.0.5/30

interface eth1
 ip address 192.168.1.1/24

ip route 0.0.0.0/0 10.0.0.6

router bgp 65001
 network 192.168.1.0/24
 neighbor 10.0.0.6 remote-as 100
```

Script di init

Eseguirlo per iniziallizare FRR

`net.sh`

```
#!/bin/bash

vtysh -f frrconf
```

- **switch**

Script di init

Eseguirlo per configurare lo switch.

1. Vengono eseguiti i comandi classici per la configurazione dello switch classico con capability di VLAN filtering.
2. Vengono aggiunte le VLAN ID (95,32 tagged) per il trunk link
3. Viene aggiunta la VLAN ID 10 untagged sul trunk link (untagged)
4. Viene aggiunto un nuovo dispositivo virtuale "auth.bridge0.10" a partire dal dispositivo "bridge0", a cui viene assegnata come route di default 192.168.2.1 (il CE)
5. Con ebtables si blocca da subito ogni tipologia di forwarding, tranne per eth2.
6. Installa la conf. di hostapd

Il nuovo dispositivo virtuale "auth.bridge0.10" è necessario per permettere a hostapd di comunicare con il server RADIUS e allo stesso tempo permettere allo switch di agire da... switch, nel senso che sul link fisico eth2 devono passare sia le req/resp RADIUS che il forwarding dei frame VLAN (trunk link) verso il CE da parte dei dispositivi client.

In eth2 la PVID untagged è 10, l'interfaccia "auth.bridge0.10" è di fatto utilizzata da hostapd, ha IPv4 192.168.2.2 e route di default 192.168.2.1

I frame EAPOL devono poter passare per permettere l'autenticazione. I frame EAPOL hanno come destinazione PAE / Nearest-non-TPMR-bridge - lo switch fa da autenticatore ma non può avere un vero e proprio indirizzo MAC - su questo effettua le operazioni di switching, ma i frame di auth 802.1x/EAPOL sono diretti proprio verso lo switch che fa da authenticator.

`net.sh`

```
#!/bin/bash

ip link add bridge0 type bridge
ip link set bridge0 type bridge vlan_filtering 1
ip link set eth0 master bridge0
ip link set eth1 master bridge0
ip link set eth2 master bridge0
ip link set bridge0 up

bridge vlan del vid 1 dev eth2 pvid untagged

bridge vlan add vid 95 dev eth2
bridge vlan add vid 32 dev eth2
bridge vlan add vid 10 dev eth2 pvid untagged

bridge vlan add dev bridge0 vid 95 self
bridge vlan add dev bridge0 vid 32 self
bridge vlan add dev bridge0 vid 10 self

ip link add auth.bridge0.10 link bridge0 type vlan id 10
ip addr add 192.168.2.2/24 dev auth.bridge0.10

ip link set auth.bridge0.10 up
```

```
ip route add default via 192.168.2.1 dev auth.bridge0.10

echo 8 > /sys/class/net/bridge0/bridge/group_fwd_mask

ebtables -P FORWARD DROP
ebtables -P INPUT ACCEPT
ebtables -P OUTPUT ACCEPT
ebtables -A FORWARD -i eth2 -j ACCEPT

INSTALLDIR=/etc/hostapd
install -D -m600 hostapd.conf $INSTALLDIR/hostapd.conf
```

hostapd.sh

```
#!/bin/bash

hostapd -d /etc/hostapd/hostapd.conf
```

hostapd.conf

```
# Control interface settings
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0

# Enable logging for all modules
logger_syslog=-1
logger_stdout=-1

# Log level
logger_syslog_level=2
logger_stdout_level=2

# Driver interface type
driver=wired

# Enable IEEE 802.1X authorization
ieee8021x=1

# Use port access entry (PAE) group address
# (01:80:c2:00:00:03) when sending EAPOL frames
use_pae_group_addr=1

# Network interface for authentication requests
interface=bridge0

# Local IP address used as NAS-IP-Address
own_ip_addr=192.168.2.2
```

```
# Unique NAS-Identifier within scope of RADIUS server
nas_identifier=hostapd.example.org

# RADIUS authentication server
auth_server_addr=192.168.1.2
auth_server_port=1812
auth_server_shared_secret=mysecretpasswd
```

Programma eBPF/XDP

Nota: nel container, la directory proj3-xdp/ si trova dentro la directory /root/xdp-tutorial/

Breve spiegazione del funzionamento

Abbiamo due programmi XDP: uno per "EAPOL" e uno per "RADIUS" e il programma utente.

Il programma XDP EAPOL in realtà tiene anche conto della visibilità della stazione: nel senso che tiene traccia, una volta autenticata, quando è l'ultima volta che ha inviato un pacchetto attraverso l'interfaccia di competenza, oltre a gestire la fase iniziale di autenticazione (EAPOL/EAP). Viene installato su eth0, eth1.

Il programma XDP RADIUS parsa, sull'interfaccia che collega lo switch al router, appunto, i messaggi RADIUS, e in particolare i messaggi di accept. Viene installato su eth2.

La situazione è la seguente:

- hostapd è in esecuzione sullo switch e implementa 802.1x/EAPOL e comunica con RADIUS
- I programmi XDP sono in esecuzione sulle interfacce

Quando un client vuole autenticarsi, invia una richiesta EAPOL-Start allo switch (PAE).

Prima o poi il client deve inviare una "Response, Identity" che viene intercettata dal programma XDP EAPOL, estraendone l'identità.

Prima di continuare, è opportuno precisare che l'unico modo sicuro per correlare le richieste EAPOL alle risposte di Accept di RADIUS è tramite l'identità a causa del fatto che lo switch forwarda su tutte le porte le richieste EAPOL: questo significa che tutti i client che hanno in esecuzione wpa_supplicant inizieranno un processo di auth con stesso ID EAP dell'originale. Usare l'ID EAP nei messaggi EAP incapsulati da RADIUS, non è quindi possibile, per comprendere se l'auth è avvenuta con successo.

L'identità è sicura da usare:

- I messaggi RADIUS Access-Accept contengono l'identità
- una sola stazione ha l'identità specifica
- una sola stazione su una sola interfaccia può usare quella identità

Il messaggio RADIUS Access-Reject viene ignorato, finchè non si riscontra un messaggio RADIUS Access-Accept corrispondente a quello EAPOL (sempre matchando l'identità), l'autenticazione della stazione rimane

pendente, dopo un tot di tempo che la richiesta di auth rimane pendente (1 minuto), quest'ultima viene eliminata.

E' accettabile che vengano fatte ulteriori richieste, che sostituiscono la precedente, pendente, solo dalla stessa interfaccia da cui è stata generata quest'ultima.

Una volta che il messaggio Access-Accept viene riscontrato, si elimina la entry nella mappa dell'auth pending, se ne crea una nuova in quella delle stazioni autenticate e ogni ulteriore richiesta di autenticazione della stazione autenticata viene ignorata. Dal messaggio RADIUS Access-Accept viene estratto il VLAN ID e inserito nella entry della stazione autenticata (finalizzazione dell'auth. da parte del programma XDP RADIUS).

Il programma user effettua polling sulla mappa delle stazioni autenticate e rileva una nuova entry, che ancora non ha gestito. Quindi esegue bridge vlan e ebtables per permettere l'accesso alla LAN alla stazione e assegnargli un VLAN ID: dalla mappa delle stazioni autenticate c'è la VLAN ID e il MAC della stazione, quindi flagga l'entry come "user_known", in modo da sapere che l'entry è stata gestita.

Ogni 5 secondi il programma userspace effettua una scansione lineare sulla mappa e capisce cosa deve fare.

Inoltre, il programma userspace viene "informato" dal programma XDP (sempre tramite polling, aggiornando entry nella mappa eBPF):

- Se la stazione ha inviato il frame EAPOL-Logoff
- Quanto tempo fa la stazione ha trasmesso l'ultimo pacchetto
- Se la stazione ha cambiato interfaccia

Quando il programma XDP EAPOL sulle interfacce eth0, eth1 rilevano la prima o la terza condizione, bloccano il traffico (XDP_DROP) per permettere all'user program di agire: impedire l'accesso alla rete in modo definitivo con ebtables e bridge vlan e rimuovere la entry dalla mappa eBPF.

Il secondo caso è completamente a discrezione del programma user - il programma XDP segnala solamente quand'è che ha visto l'ultima volta pacchetti sull'interfaccia da quella stazione, l'user decide quando reputare la stazione disconnessa, e agire di conseguenza come nel punto precedente.

In realtà, di default il supporto per il rilevamento del fatto che la stazione ha cambiato interfaccia è acerbo, può dare problemi se si scambiano le interfacce delle due stazioni in certi interleaving, ma in più è stato implementato un supporto migliore (comunque "sperimentale"), che si può abilitare decommentando la linea "//#define EXTENDED_SUPPORT" nel file sorgente C del programma utente.

Di default (ovvero senza EXTENDED_SUPPORT) l' "allow" da parte del programma utente consiste nell'aggiungere la regola -A FORWARD -s {mac} -j ACCEPT a ebtables, assegnare il PVID (estratto dal messaggio RADIUS) untagged con bridge vlan sull'interfaccia, e il deny consiste in una regola -D FORWARD -s {mac} -j ACCEPT con ebtables (eliminando la regola installata prima) e il ripristino del PVID untagged 1 sull'interfaccia, eliminando quello installato prima (è diverso per quanto riguarda EXTENDED_SUPPORT, ma è più di interesse la modalità di "default").

Breve guida all'utilizzo

- Eseguire lo script "/root/net.sh"
- Da tmux, aprire due terminali, in uno, eseguire "/root/hostapd.sh"

- Nell'altro terminale, entrare in "/root/xdp-tutorial/proj3-xdp" (è necessario che "proj3-xdp" sia in "xdp-tutorial" perchè fa affidamento sui suoi Makefile)
- Eseguire make clean
- Eseguire make
- Eseguire make install-xdp (utilizza direttamente ip link per il caricamento e l'attach del programma XDP)
- Eseguire ./xdp_prog_user

Quando non è più necessario,

- Terminare il programma utente con CTRL+C
- Eseguire make remove-xdp per disinstallare il programma XDP e rimuovere le mappe pinnate

Header di configurazione

`proj3-xdp/config.h`

```
#ifndef CONFIG_H
#define CONFIG_H

/*
 * set of configurable params
 */

#define CONFIG_MAX_CONN_STAS 2
#define CONFIG_MAX_IDENT_NAME_LEN 20
#define CONFIG_MAX_IDENTS 2
#define CONFIG_PENDING_AUTH_DISCARD_NS 60000000000
#define CONFIG_RADIUS_SPORT 1812
#define CONFIG_RADIUS_MAX_AVPS 200

#endif
```

Header di macro __die

`proj3-xdp/die.h`

```
#ifndef DIE_H
#define DIE_H

#define __die0(fun) \
    do { \
        perror(#fun); \
        exit(EXIT_FAILURE); \
    } while(0)
```

```
#define __die1(fun, param) \
    do { \
        fprintf(stderr, #fun "(%s): %s\n", param, strerror(errno)); \
        exit(EXIT_FAILURE); \
    } while(0)

#endif
```

Header di defs per EAPOL

proj3-xdp/eapol.h

```
#ifndef EAPOL_H
#define EAPOL_H

struct eapolhdr {
    __u8 version;
    __u8 type;
    __u16 length;
} __attribute__((packed));

struct eaphdr {
    __u8 code;
    __u8 id;
    __u16 length;
} __attribute__((packed));

struct eapdata {
    __u8 type;
} __attribute__((packed));

#define EAP_RESPONSE 2
#define EAP_RESPONSE_TYPE_IDENTITY 1
#define EAPOL_EAP 0
#define EAPOL_LOGOFF 2
#define HAS_EAPOL(frame) ((frame)->h_proto == bpf_htons(ETH_P_PAE))

#endif
```

Header per le mappe

proj3-xdp/maps.h

```
#ifndef MAPS_H
#define MAPS_H

#ifndef CONFIG_H
#error You must include config.h before maps.h
```

```
#endif

struct pending_auth_sta_val {
    __u64 req_issue_time;
    __u32 iface;
    __u8 macaddr[6];
};

struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, CONFIG_MAX_IDENTS);
    __type(key, __u8[CONFIG_MAX_IDENT_NAME_LEN]);
    __type(value, struct pending_auth_sta_val);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
} pending_auth_sta SEC(".maps");

#include "shmapsdefs.h"

struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, CONFIG_MAX_CONN_STAS);
    __type(key, __u8[6]);
    __type(value, struct authd_sta_val);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
} authd_sta SEC(".maps");

#endif
```

Header per le defs. RADIUS

`proj3-xdp/radius.h`

```
#ifndef RADIUS_H
#define RADIUS_H

#ifndef CONFIG_H
#error You must include config.h before radius.h
#endif

// note : length includes # of bytes for the whole RADIUS payload within UDP
//        that is - code + id + length + authenticator + avps

struct radiushdr {
    __u8 code;
    __u8 id;
    __u16 length;
    __u8 authenticator[16];
} __attribute__((packed));

#define RADIUS_CODE_ACCESS_ACCEPT 2
```

```
// note : length includes # of bytes for the whole AVP payload within RADIUS
//        that is - avp.type + avp.length + avp.type_specific_data

struct radiusavphdr {
    __u8 type;
    __u8 length;
} __attribute__((packed));

#define RADIUS_AVP_TYPE_USER_NAME 1
#define RADIUS_AVP_TYPE_TUNNEL_PRIVATE_GROUP_ID 81

#define HAS_IP(_eth) ((_eth)->h_proto == bpf_htons(ETH_P_IP))
#define HAS_UDP(_ip) ((_ip)->protocol == IPPROTO_UDP)
#define HAS_RADIUS(_udp) ((_udp)->source == bpf_htons(CONFIG_RADIUS_SPORT))

#endif
```

Header per value mappa utilizzato anche dal programma user

`proj3-xdp/shmapsdefs.h`

```
#ifndef SHMAPSDEFS_H
#define SHMAPSDEFS_H

struct authd_sta_val {
    __u64 last_seen;
    __u32 current_iface;
    __u32 origin_iface;
    __u8 vlan_id[5];
    __u8 user_known;
    __u8 supplicant_logoff;
};

#endif
```

Sorgente C programma XDP EAPOL

`proj3-xdp/xdp_prog_kern_eapol.c`

```
#include <linux/bpf.h>
#include <linux/if_ether.h>

#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>

#include "config.h"
#include "maps.h"
#include "eapol.h"
```

```
static long __check_req_issue_time_cb(void *map, const void *key, const void
*value, void *ctx) {
    struct pending_auth_sta *psta = (struct pending_auth_sta*) map;
    struct pending_auth_sta_val *psta_val = (struct pending_auth_sta_val*) value;
    __u8 *psta_key = (__u8*) key;

    __u64 now_time = *((__u64*) ctx);

    if(now_time - psta_val->req_issue_time >= CONFIG_PENDING_AUTH_DISCARD_NS) {
        if(bpf_map_delete_elem(psta, psta_key) < 0) {
            bpf_printk("unable to delete pending request (bpf_map_delete_elem
failure)\n");
        }
    }

    return 0;
}

static int __do_start_auth(__u8* macaddr, __u32 iface, struct eapdata *data, __u16
typedatalen, void* data_end) {
    if(typedatalen > CONFIG_MAX_IDENT_NAME_LEN) {
        return XDP_DROP;
    }

    if(((void*)data) + sizeof(struct eapdata) > data_end) {
        return XDP_DROP;
    }

    if(data->type != EAP_RESPONSE_TYPE_IDENTITY) {
        return XDP_PASS;
    }

    __u64 now = bpf_ktime_get_boot_ns();
    bpf_for_each_map_elem(&pending_auth_sta, __check_req_issue_time_cb, &now, 0);

    __u8 *pkt_identity = (__u8*)(((void*)data) + sizeof(struct eapdata));

    __u8 identity[CONFIG_MAX_IDENT_NAME_LEN];
    __builtin_memset(identity, 0, CONFIG_MAX_IDENT_NAME_LEN);
    if(bpf_probe_read_kernel(identity, typedatalen, pkt_identity) < 0) {
        return XDP_DROP;
    }

    struct pending_auth_sta_val *psta_val = bpf_map_lookup_elem(&pending_auth_sta,
identity);
    if(psta_val != NULL && psta_val->iface != iface) {
        return XDP_PASS;
    }

    struct pending_auth_sta_val new_psta_val;
    __builtin_memset(&new_psta_val, 0, sizeof(struct pending_auth_sta_val));

    new_psta_val.req_issue_time = now;
    __builtin_memcpy(new_psta_val.macaddr, macaddr, sizeof(__u8) * 6);
```

```
    new_psta_val.iface = iface;

    if(bpf_map_update_elem(&pending_auth_sta, identity, &new_psta_val, BPF_ANY) <
0) {
        return XDP_DROP;
    }

    return XDP_PASS;
}

static int check_if_supplicant_logoff(struct ethhdr* frame, struct authd_sta_val
*sta, void *data_end) {
    if(HAS_EAPOL(frame)) {
        struct eapolhdr *eapol = (struct eapolhdr*) (((void*)frame) +
sizeof(struct ethhdr));
        if(((void*)eapol) + sizeof(struct eapolhdr) > data_end) {
            return XDP_DROP;
        }

        if(eapol->type == EAPOL_LOGOFF) {
            sta->supplicant_logoff = 1;
        }
    }

    return XDP_PASS;
}

static int attempt_start_auth(__u32 iface, struct ethhdr* frame, void* data_end) {
    if(HAS_EAPOL(frame)) {
        struct eapolhdr *eapol = (struct eapolhdr*) (((void*)frame) +
sizeof(struct ethhdr));
        if(((void*)eapol) + sizeof(struct eapolhdr) > data_end) {
            return XDP_DROP;
        }

        if(eapol->type == EAPOL_EAP) {
            struct eaphdr *eap = (struct eaphdr*) (((void*)eapol) + sizeof(struct
eapolhdr));
            if(((void*)eap) + sizeof(struct eaphdr) > data_end) {
                return XDP_DROP;
            }

            if(eap->code == EAP_RESPONSE) {
                struct eapdata *eapdata = (struct eapdata*) (((void*)eap) +
sizeof(struct eaphdr));
                __u16 tdlen = bpf_ntohs(eap->length) - sizeof(struct eapdata) -
sizeof(struct eaphdr);
                return __do_start_auth(frame->h_source, iface, eapdata, tdlen,
data_end);
            }
        }
    }

    return XDP_PASS;
```

```
}

SEC("xdp")
int inspect_eapol_frame(struct xdp_md* ctx) {
    void *data_end = (void*)((__u64) ctx->data_end);
    void *data = (void*)((__u64) ctx->data);

    struct ethhdr *eth = data;
    if(((void*)eth) + sizeof(struct ethhdr) > data_end) {
        return XDP_DROP;
    }

    struct authd_sta_val *sta = bpf_map_lookup_elem(&authd_sta, eth->h_source);
    if(sta != NULL) {
        __u32 sta_cur_iface = sta->current_iface;
        __u32 sta_orig_iface = sta->origin_iface;
        __u8 sta_logoff = sta->supplicant_logoff;

        if(sta_cur_iface != sta_orig_iface || sta_logoff) {
            return XDP_DROP;
        }

        sta->current_iface = ctx->ingress_ifindex;

        if(sta->current_iface != sta_orig_iface) {
            return XDP_DROP;
        }

        sta->last_seen = bpf_ktime_get_boot_ns();

        return check_if_supplicant_logoff(eth, sta, data_end);
    }

    return attempt_start_auth(ctx->ingress_ifindex, eth, data_end);
}

char LICENSE[] SEC("license") = "GPL";
```

Sorgente C programma XDP RADIUS

`proj3-xdp/xdp_prog_kern_radius.c`

```
#include <linux/bpf.h>
#include <linux/if_ether.h>
#include <linux/ip.h>
#include <linux/in.h>
#include <linux/udp.h>

#include <bpf/bpf_helpers.h>
#include <bpf/bpf_endian.h>
```

```
#include "config.h"
#include "maps.h"
#include "radius.h"

static int parse_radius_avps(struct radiusavphdr *avps, __u8* username, __u8* vid,
void* data_end) {
    __u8 has_vid = 0;
    __u8 has_username = 0;

    for(int i = 0; i < CONFIG_RADIUS_MAX_AVPS; i++) {
        if(((void*)avps) + sizeof(struct radiusavphdr) > data_end) {
            return XDP_DROP;
        }

        if(avps->length < 2) {
            return XDP_DROP;
        }

        if(((void*)avps) + avps->length > data_end) {
            return XDP_DROP;
        }

        __u8 *data = ((void*)avps) + sizeof(struct radiusavphdr);
        __u16 datalen = avps->length - sizeof(struct radiusavphdr);

        if(avps->type == RADIUS_AVP_TYPE_USER_NAME) {
            if(datalen > CONFIG_MAX_IDENT_NAME_LEN) {
                return XDP_DROP;
            }

            if(bpf_probe_read_kernel(username, datalen, data) < 0) {
                return XDP_DROP;
            }

            has_username = 1;
        } else if(avps->type == RADIUS_AVP_TYPE_TUNNEL_PRIVATE_GROUP_ID) {
            if(datalen > 4) {
                return XDP_DROP;
            }

            if(bpf_probe_read_kernel(vid, datalen, data) < 0) {
                return XDP_DROP;
            }

            has_vid = 1;
        }

        if(has_username && has_vid) {
            return XDP_PASS;
        }

        avps = ((void*)avps) + avps->length;
    }
```

```
    return XDP_PASS;
}

static int finalize_auth(__u8 *identity, __u8 *vid) {
    struct pending_auth_sta_val *pendauthsta =
bpf_map_lookup_elem(&pending_auth_sta, identity);
    if(pendauthsta == NULL) {
        return XDP_PASS;
    }

    struct authd_sta_val authdsta;
    __builtin_memset(&authdsta, 0, sizeof(struct authd_sta_val));
    authdsta.last_seen = bpf_ktime_get_boot_ns();
    authdsta.current_iface = pendauthsta->iface;
    authdsta.origin_iface = pendauthsta->iface;
    __builtin_memcpy(authdsta.vlan_id, vid, 5);
    authdsta.user_known = 0;
    authdsta.supplicant_logoff = 0;

    __u8 macaddr[6];
    __builtin_memcpy(macaddr, pendauthsta->macaddr, 6);

    if(bpf_map_delete_elem(&pending_auth_sta, identity) < 0) {
        return XDP_DROP;
    }

    if(bpf_map_update_elem(&authd_sta, macaddr, &authdsta, BPF_NOEXIST) < 0) {
        return XDP_DROP;
    }

    return XDP_PASS;
}

static int extract_radiushdr(struct ethhdr *frame, struct radiushdr **out, void
*data_end) {
    *out = NULL;

    if(HAS_IP(frame)) {
        struct iphdr *ip = (struct iphdr*) (((void*)frame) + sizeof(struct
ethhdr));
        if(((void*)ip) + sizeof(struct iphdr) > data_end) {
            return XDP_DROP;
        }

        if(HAS_UDP(ip)) {
            struct udphdr *udp = (struct udphdr*) (((void*)ip) + sizeof(struct
iphdr));
            if(((void*)udp) + sizeof(struct udphdr) > data_end) {
                return XDP_DROP;
            }

            if(HAS_RADIUS(udp)) {
                struct radiushdr *radius = (struct radiushdr*) (((void*)udp) +
```

```
sizeof(struct udphdr));
                if(((void*)radius) + sizeof(struct radiushdr) > data_end) {
                    return XDP_DROP;
                }

                *out = radius;
            }
        }
    }

    return XDP_PASS;
}

SEC("xdp")
int inspect_radius_frame(struct xdp_md* ctx) {
    void *data_end = (void*)((__u64) ctx->data_end);
    void *data = (void*)((__u64) ctx->data);

    struct ethhdr *eth = data;
    if(((void*)eth) + sizeof(struct ethhdr) > data_end) {
        return XDP_DROP;
    }

    struct radiushdr *radius;
    int rv = extract_radiushdr(eth, &radius, data_end);
    if(radius == NULL) {
        return rv;
    }

    if(radius->code == RADIUS_CODE_ACCESS_ACCEPT) {
        struct radiusavphdr *avps = (struct radiusavphdr*) (((void*)radius) +
sizeof(struct radiushdr));

        __u8 username[CONFIG_MAX_IDENT_NAME_LEN];
        __u8 vid[5];

        __builtin_memset(username, 0, CONFIG_MAX_IDENT_NAME_LEN);
        __builtin_memset(vid, 0, 5);

        rv = parse_radius_avps(avps, username, vid, data_end);
        if(username[0] == 0 || vid[0] == 0) {
            return rv;
        }

        return finalize_auth(username, vid);
    }

    return XDP_PASS;
}

char LICENSE[] SEC("license") = "GPL";
```

Sorgente C programma utente

proj3-xdp/xdp_prog_user.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <linux/limits.h>
#include <bpf/bpf.h>
#include <net/if.h>

#include "die.h"
#include "shmapsdefs.h"

//#define EXTENDED_SUPPORT

#ifdef EXTENDED_SUPPORT
#warning extended support is enabled
#endif

#define NANO_PER_SEC 1000000000LL

#ifndef DEFAULT_DISCONNECT_AFTER_INACTIVITY_THR_SEC
#define DEFAULT_DISCONNECT_AFTER_INACTIVITY_THR_SEC (10 * 60)
#endif

#ifndef DEFAULT_BPFFS_PATH
#define DEFAULT_BPFFS_PATH "/sys/fs/bpf/xdp/globals"
#endif

#ifndef DEFAULT_EBPF_MAP
#define DEFAULT_EBPF_MAP "authd_sta"
#endif

extern int optind;
extern char* optarg;

static char bpffs[PATH_MAX];
static char ebpf_map_name[NAME_MAX];
static __u64 disconnect_thr_ns;

static int open_bpf_map_by_name() {
    char full_map_path[PATH_MAX];
    memset(full_map_path, 0, sizeof(full_map_path));

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wformat-truncation"
    snprintf(full_map_path, PATH_MAX, "%s/%s", bpffs, ebpf_map_name);
#pragma GCC diagnostic pop

    int fd = bpf_obj_get(full_map_path);
```

```
    if(fd < 0) {
        __die1(bpf_obj_get, full_map_path);
    }

    return fd;
}

static void load_bpffs(const char* path) {
    memset(bpffs, 0, sizeof(bpffs));
    strncpy(bpffs, path, sizeof(bpffs) - 1);
}

static void load_ebpf_map_name(const char* map_name) {
    memset(ebpf_map_name, 0, sizeof(ebpf_map_name));
    strncpy(ebpf_map_name, map_name, sizeof(ebpf_map_name) - 1);
}

static void load_disconnect_thr_ns(__u64 thr_sec) {
    disconnect_thr_ns = thr_sec * NANO_PER_SEC;
}

static void print_program_settings() {
    printf(" * BPF fs path: %s\n", bpffs);
    printf(" * eBPF map: %s\n", ebpf_map_name);
    printf(" * Station inactivity threshold: %lld secs\n", disconnect_thr_ns /
NANO_PER_SEC);
}

static __u64 charp_to_ull(const char* s) {
    errno = 0;
    char * endp;
    __u64 v = strtoull(s, &endp, 10);

    if(errno == ERANGE) {
        printf("number %s of out of range\n", s);
        exit(EXIT_FAILURE);
    }

    if(endp == s || *endp != 0) {
        printf("invalid number %s\n", s);
        exit(EXIT_FAILURE);
    }

    return v;
}

static void event_polling(int);

int main(int argc, char **argv) {
    if(getuid() != 0) {
        fputs("this program must be run as root\n", stderr);
        return EXIT_FAILURE;
    }
```

```
    load_bpffs(DEFAULT_BPFFS_PATH);
    load_ebpf_map_name(DEFAULT_EBPF_MAP);
    load_disconnect_thr_ns(DEFAULT_DISCONNECT_AFTER_INACTIVITY_THR_SEC);

    char optch;
    while((optch = getopt(argc, argv, "p:m:t:")) != -1) {
        if(optch == 'p') {
            load_bpffs(optarg);
        } else if(optch == 'm') {
            load_ebpf_map_name(optarg);
        } else if(optch == 't') {
            load_disconnect_thr_ns(charp_to_ull(optarg));
        }
    }

    print_program_settings();

    int map_fd = open_bpf_map_by_name();
    event_polling(map_fd);

    //unreachable code
    close(map_fd);
    return EXIT_SUCCESS;
}

/* Event polling and policy enforcement */

#define timespec_to_ns(ts) (((__u64)(ts).tv_sec) * NANO_PER_SEC + (ts).tv_nsec)

#define NEED_TO_DENY_ACCESS(x) \
    ((subtract_times(timespec_to_ns(now), (x).last_seen) > disconnect_thr_ns) || \
    ((x).current_iface != (x).origin_iface) || \
    ((x).supplicant_logoff))

#define subtract_times(_ns_x, _ns_y) \
    ((_ns_y) >= (_ns_x) ? 0 : (_ns_x) - (_ns_y))

#define arrcmp(_x, _y, _sz) ({ \
    int rv = 1; \
    for(int i = 0; i < _sz; i++) { \
        if(_x[i] != _y[i]) { \
            rv = 0; \
            break; \
        } \
    } \
    rv; })

#define DEFINE_IFNAME_FROM_IFINDEX(_ifnamevar, ifindex) \
    char _ifnamevar[IF_NAMESIZE]; \
    if(if_indextoname(ifindex, _ifnamevar) == NULL) { \
        __die0(if_indextoname); \
    }

#define DEFINE_STRFTIME(_varn) \
```

```
    char _varn[100]; \
    __fmttime(_varn)

#define CMDLINE_MAX 256

static void run_command(const char* cmd) {
    FILE *fp = popen(cmd, "r");
    if(fp == NULL) {
        __die0(popen);
    }

    int wstatus = pclose(fp);
    if(wstatus == -1) {
        __die0(pclose);
    }

    if(WIFEXITED(wstatus)) {
        if(WEXITSTATUS(wstatus) != 0) {
            fprintf(stderr, "command \"%s\" exited with exit code %d\n", cmd,
WEXITSTATUS(wstatus));
            exit(EXIT_FAILURE);
        }
    } else {
        fprintf(stderr, "command \"%s\" did *not* exit()\n", cmd);
        exit(EXIT_FAILURE);
    }
}

static void allow_access(const __u8 *macaddr, const struct authd_sta_val *val) {
    char cmdbuf[CMDLINE_MAX];
    DEFINE_IFNAME_FROM_IFINDEX(ifname, val->origin_iface);

#ifdef EXTENDED_SUPPORT
    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "bridge vlan del dev %s vid 95 pvid untagged",
ifname);
    run_command(cmdbuf);

    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "bridge vlan del dev %s vid 32 pvid untagged",
ifname);
    run_command(cmdbuf);
#endif

    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "bridge vlan add dev %s vid %s pvid untagged",
ifname, val->vlan_id);
    run_command(cmdbuf);

    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "ebtables -A FORWARD -s
%02X:%02X:%02X:%02X:%02X:%02X -j ACCEPT",
        macaddr[0], macaddr[1], macaddr[2], macaddr[3], macaddr[4], macaddr[5]);
    run_command(cmdbuf);
```

```
}

static void deny_access(const __u8 *macaddr, const struct authd_sta_val *val) {
    char cmdbuf[CMDLINE_MAX];
    DEFINE_IFNAME_FROM_IFINDEX(ifname, val->origin_iface);

#ifndef EXTENDED_SUPPORT
    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "bridge vlan del dev %s vid %s pvid untagged",
ifname, val->vlan_id);
    run_command(cmdbuf);

    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "bridge vlan add dev %s vid 1 pvid untagged",
ifname);
    run_command(cmdbuf);
#endif

    memset(cmdbuf, 0, CMDLINE_MAX);
    snprintf(cmdbuf, CMDLINE_MAX, "ebtables -D FORWARD -s
%02X:%02X:%02X:%02X:%02X:%02X -j ACCEPT",
        macaddr[0], macaddr[1], macaddr[2], macaddr[3], macaddr[4], macaddr[5]);
    run_command(cmdbuf);
}

static void __fmttime(char buf[100]) {
    memset(buf, 0, 100);

    time_t raw;
    time(&raw);

    struct tm *timeinfo = localtime(&raw);
    strftime(buf, 100, "%F %r", timeinfo);
}

static void __base_log(const char* evt, const __u8 *macaddr, const struct
authd_sta_val *val) {
    DEFINE_STRFTIME(nowtime);
    DEFINE_IFNAME_FROM_IFINDEX(curiface, val->current_iface);
    DEFINE_IFNAME_FROM_IFINDEX(origiface, val->origin_iface);

    printf(
            "[%s] %s access for %02X:%02X:%02X:%02X:%02X:%02X with vid %s,"
            " origin iface: %s, current iface: %s,"
            " logoff: %s\n",
        nowtime, evt,
        macaddr[0], macaddr[1], macaddr[2], macaddr[3],
        macaddr[4], macaddr[5], val->vlan_id,
        origiface, curiface,
        val->supplicant_logoff ? "yes" : "no");
}

static void log_access_allowed(const __u8 *macaddr, const struct authd_sta_val
*val) {
```

```
    __base_log("allowed", macaddr, val);
}

static void log_access_denied(const __u8 *macaddr, const struct authd_sta_val
*val) {
    __base_log("denied", macaddr, val);
}

#ifdef EXTENDED_SUPPORT

static void deny_access_to_sta_on_iface(int map, __u32 ifindex, __u8 *notkey) {
    __u8 prev_key[6];
    __u8 cur_key[6];
    void *prev_key_ptr = NULL;

    int rv;
    while((rv = bpf_map_get_next_key(map, prev_key_ptr, cur_key)) == 0) {
        struct authd_sta_val cur_val;

        if(bpf_map_lookup_elem(map, cur_key, &cur_val) != 0) {
            __die0(bpf_map_lookup_elem);
        }

        if(!arrcmp(notkey, cur_key, 6) && cur_val.current_iface == ifindex &&
cur_val.user_known) {
            deny_access(cur_key, &cur_val);
            if(bpf_map_delete_elem(map, cur_key) < 0) {
                __die0(bpf_map_delete_elem);
            }

            log_access_denied(cur_key, &cur_val);
            return;
        }

        memcpy(prev_key, cur_key, sizeof(__u8) * 6);
        if(prev_key_ptr == NULL) {
            prev_key_ptr = prev_key;
        }
    }

    if(rv != -ENOENT) {
        __die0(bpf_map_get_next_key);
    }
}

#endif

static void event_polling(int map) {
    printf("starting event polling...\n");

    while(1) {
        struct timespec now;

        if(clock_gettime(CLOCK_BOOTTIME, &now) < 0) {
```

```
            __die0(clock_getttime);
        }

        __u8 prev_key[6];
        __u8 cur_key[6];
        void *prev_key_ptr = NULL;

        int rv;
        while((rv = bpf_map_get_next_key(map, prev_key_ptr, cur_key)) == 0) {
            struct authd_sta_val cur_val;

            if(bpf_map_lookup_elem(map, cur_key, &cur_val) != 0) {
                __die0(bpf_map_lookup_elem);
            }

            if(NEED_TO_DENY_ACCESS(cur_val)) {
#ifdef EXTENDED_SUPPORT
                if(cur_val.origin_iface != cur_val.current_iface) {
                    deny_access_to_sta_on_iface(map, cur_val.current_iface,
cur_key);
                }
#endif
                deny_access(cur_key, &cur_val);
                if(bpf_map_delete_elem(map, cur_key) < 0) {
                    __die0(bpf_map_delete_elem);
                }

                log_access_denied(cur_key, &cur_val);
                continue;
            }

            if(!cur_val.user_known) {
#ifdef EXTENDED_SUPPORT
                deny_access_to_sta_on_iface(map, cur_val.current_iface, cur_key);
#endif
                allow_access(cur_key, &cur_val);
                cur_val.user_known = 1;
                if(bpf_map_update_elem(map, cur_key, &cur_val, 0) < 0) {
                    __die0(bpf_map_update_elem);
                }

                log_access_allowed(cur_key, &cur_val);
            }

            memcpy(prev_key, cur_key, sizeof(__u8) * 6);
            if(prev_key_ptr == NULL) {
                prev_key_ptr = prev_key;
            }
        }

        if(rv != -ENOENT) {
            __die0(bpf_map_get_next_key);
        }
```

```
        sleep(5);
    }
}
```

Makefile

proj3-xdp/Makefile

```
# SPDX-License-Identifier: (GPL-2.0 OR BSD-2-Clause)

XDP_PROG_KERN_EAPOL = xdp_prog_kern_eapol
XDP_PROG_KERN_RADIUS = xdp_prog_kern_radius
XDP_PROG_USER = xdp_prog_user

XDP_TARGETS  := $(XDP_PROG_KERN_EAPOL) $(XDP_PROG_KERN_RADIUS)
USER_TARGETS := $(XDP_PROG_USER)

LIBBPF_DIR = ../libbpf/src/
COMMON_DIR = ../common

EXTRA_DEPS := $(COMMON_DIR)/parsing_helpers.h

include $(COMMON_DIR)/common.mk

# ---------------------------------
# ---- ste's install commands -----
# ---------------------------------

SHELL=/bin/bash

TARGET_IFACES_RADIUS ?= eth2

install-xdp-radius: $(XDP_PROG_KERN_RADIUS).o
        @for iface in $(TARGET_IFACES_RADIUS); do \
                ip link set dev $${iface} xdp obj $(XDP_PROG_KERN_RADIUS).o sec
xdp && \
                        echo "[ OK ] RADIUS xdp program installed for $${iface}"
|| \
                        echo "[ FAIL ] RADIUS xdp program **NOT** installed for
$${iface}"; \
        done

TARGET_IFACES_EAPOL ?= eth0 eth1

install-xdp-eapol: $(XDP_PROG_KERN_EAPOL).o
        @for iface in $(TARGET_IFACES_EAPOL); do \
                ip link set dev $${iface} xdp obj $(XDP_PROG_KERN_EAPOL).o sec xdp
&& \
                        echo "[ OK ] EAPOL xdp program installed for $${iface}" ||
\
                        echo "[ FAIL ] EAPOL xdp program **NOT** installed for
```

```
$${iface}"; \
        done

install-xdp: install-xdp-eapol install-xdp-radius

RM_TARGET_IFACES ?= $(TARGET_IFACES_RADIUS) $(TARGET_IFACES_EAPOL)

remove-xdp-only:
        @for iface in $(RM_TARGET_IFACES); do \
                ip link set $${iface} xdp off && \
                        echo "[ OK ] xdp program removed for $${iface}" || \
                        echo "[ FAIL ] xdp program **NOT** removed for $${iface}";
\
        done

PINNED_MAPS_PATH ?= /sys/fs/bpf/xdp/globals
RM_TARGET_MAPS ?= authd_sta pending_auth_sta

remove-pinned-maps:
        @for map in $(RM_TARGET_MAPS); do \
                rm $(PINNED_MAPS_PATH)/$${map} && \
                        echo "[ OK ] map $(PINNED_MAPS_PATH)/$${map} removed" || \
                        echo "[ FAIL ] map $(PINNED_MAPS_PATH)/$${map} **NOT**
removed"; \
        done

remove-xdp: remove-xdp-only remove-pinned-maps

reinstall:
        @make remove-xdp
        @make install-xdp

deep-reinstall:
        @make clean
        @make reinstall
```

- **client-B1**

Script di init

Eseguirlo per configurare IPv4 e copiare file di config di wpa_supplicant

net.sh

```
#!/bin/bash

IPADDR=192.168.2.6/30
GATEWAY=192.168.2.5

#IPADDR=192.168.3.2/24
#GATEWAY=192.168.3.1
```

```
ip addr add $IPADDR dev eth0
ip route add default via $GATEWAY dev eth0

INSTALLDIR=/etc
install -D -m400 wpa_supplicant.conf $INSTALLDIR/wpa_supplicant.conf
```

`wpa_supplicant.conf`

```
ap_scan=0
network={
 key_mgmt=IEEE8021X
 eap=MD5
 identity="clientb1"
 password="clientb1passwd"
 eapol_flags=0
}
```

`wpa_supplicant.sh`

```
#!/bin/bash

wpa_supplicant -B -c/etc/wpa_supplicant.conf -Dwired -ieth0
```

- **client-B2**

Script di init

Eseguirlo per configurare IPv4 e copiare file di config di wpa_supplicant

`net.sh`

```
#!/bin/bash

IPADDR=192.168.2.10/30
GATEWAY=192.168.2.9

#IPADDR=192.168.4.2/24
#GATEWAY=192.168.4.1

ip addr add $IPADDR dev eth0
ip route add default via $GATEWAY dev eth0

INSTALLDIR=/etc
install -D -m400 wpa_supplicant.conf $INSTALLDIR/wpa_supplicant.conf
```

wpa_supplicant.conf

```
ap_scan=0
network={
 key_mgmt=IEEE8021X
 eap=MD5
 identity="clientb2"
 password="clientb2passwd"
 eapol_flags=0
}
```

wpa_supplicant.sh

```
#!/bin/bash

wpa_supplicant -B -c/etc/wpa_supplicant.conf -Dwired -ieth0
```