**Retele**

**Lab2**

ls | grep 234 – cauta toate fisierele care au in nume 234

intram in folderul docker

docker build -t 234 .

. ca sa acceseze fisierul Dockerfile

Daca eram in computer-networks apelam ./dockerdocker image ls

Docker image ls – verificam daca s-a pus 234

Docker run -ti 234 echo “salut” -> putem folosi -ti sau -it care sunt flaguri

Docker ps -> dupa ce a afisat salut, containerul s-a inchis

Docker run -ti 234 bash ->rulam un shell

* Rulam in containerul de docker in rtuser creat in Dockerfile

Docker-compose

Fisierul docker-compose.yml

Volumes -> **. : /elocal** -> in folderul . ( folderol current – computer-networks )

* + - * Face fisierul elocal

Avem 3 containere care ruleaza aceasi imagine, au acelasi volum

Dmz – reteaua

Docker-compose up -d

Ruleaza fisierul docker-compose.yml si face containerele declarate acolo

Docker-compose ps -> cele 3 servicii vreate ruleaza

Tty: true -> e pus ca sa porneasca sis a ruleze in continuare, sa nu se inchida

Docker-compose exec rt1 bash -> rulam primul container si se deschide in userul lui

In rtuser : ls /elocal -> toate fisierele din computer-networks

Exit -> iesim din container

Docker-compose down -> inchide si sterge containerele

**Lab 4**

Intram in folderul computer\_networks din github

Rulam:

docker image

Vrem sa instalam ceva pe imaginea grupei 234

Trebuie sa modificam in Dockerfile din folderul dockerfile

Stergem tot de la root useradd in jos, inclusive

Wireshark – urmareste pachetele care umbla pe retea -> are interfata

RUN apt-get update && apt-get install -y tcpdump wireshark curl

RUN mv usr/sbin/tcpdump /usr/local/bin

ENV PATH = "/usr/local/bin:${PATH}"

PATH -> lista de pathuri

Terminal:

docker build -t 234w ./docker

schimbam in docker-compose.yml la services imagea cu 234w

Terminal:

docker-compose up -d

docker compose exec rt1 bash

Dam in root ping rt2

Dam ping -s 2000 rt2

Iesim din ping cu ctrl + c

Cmd + shift + t -> nou terminal

Docker-compose exec rt2 bash

In root tcpdump -ntvS -> urmarim pachetele din rt2

**Lab 5**

Script.py

Import sys

Import logging

logging.basicConfig(format = u'[LINE:%(lineno)d]# %(levelname)-8s [%(asctime)s] %(message)s', level = logging.NOTSET)

cream scipt.py in computer-networks:

In terminal:

Python script.py cevaaaa

./docker-compose up -d

./docker-compose exec rt1 bash

/elocal/script.py saluut

Exit

----🡪 scriem in docker-compose.yml: la rt1:image ----🡪 un rand mai jos scriem: command: python /elocal/script.py bunaaaa

./docker-compose down

./docker-compose up -d

./docker-compose logs

./docker-compose ps

**Lab 6**

Intram pe moodle, punem un cont si o parola gresita, dam F12 (pe mac in chrome) si ne uitam la networks, la apeluri sa vedem ce tip au (post, get).

Intram in computer-networks si pornim containere

docker-compose up -d ( pe linux ai ./ in fata)

Rulam: docker-compose exec rt2 bash -> daca apare rtuser Nu e bine ( nu are tcpdump? ), trebuie root

Daca apare rtuser

* Exit
* docker images -> daca apare o grupa cu w

mai deschidem un terminal in care rulam: docker-compose exec rt2 bash

intr-un terminal scriem: tcpdump -SntveXX

in celalalt terminal: wget <http://moodle.fmi.unibuc.ro>

mergem sus de tot si ne uitam acolo unde e fffff -> trimite la toata lumea un mesaj “da-mi mac-ul tau”

Reply -> care e mac-ul gate-ului

Ne uitam mai jos unde avem: GET / HTTP/1.1

193.226.51.9 -> IP moodle -> port http 80 (mereu pt moodle) ( port https 443 )

172.18.0.3.38140 > 193.226.51.9.80

^

|

Portul : 38140

Request la un https ( chrome)

wget <https://google.com>

216.58.206.4.443

^- https - codificare la nivelul aplicatiei, nu se vede html-ul dar se vad mac-urile si porturile

UDP – user data (ceva) protocol

Port pe 16 biti – max 65535 si variaza ( 65535 – 2^16-1)

|  |  |  |  |
| --- | --- | --- | --- |
| 8 biti | 8 biti | 8 biti | 8 biti |
| Port S | | Port D | |
| Length | | Checksum | |

|  |  |
| --- | --- |
| App | HTTP |
| 44450 | 80 |
| L | C = 300 |

Tr UDP

* payloan

Ne folosim de suma pt a vedea daca pachetele ajung bine

Checksum e optional

Portul sursa e optional ( primeste un mesaj portul final dar nu poate raspunde )

Length-ul intregului pachet – 64 biti dar e scris in bytes – 8 + cati bytes are getul

Scriem in python un server

2 fisiere: server.py si client.py

In server.py:

import socket

|> IPV4

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM ) -> am importat un obiect de tip socket

adr='127.0.0.3' -> adresa de local host

port = 10000

adr\_server = (adr,port) # -> tuplu

in server

s.bind(adr\_server) -> aloca portul – leaga adresa de port

(m,a) = s.recvfrom(4) -> cu recvfrom citeste mesaje din buffer daca exista, daca nu exista asteapta sa primeasca un mesaj / 4 – cati byte sa citeasca - daca dau 2 o sa citeasca doar he

print(m,a)

s.sendto("salut".encode("utf-8"), a)

s.close()

in client

|> un character = un byte

s.sendto("hey".encode("utf-8"),adr\_server) -> codeaza mesajul si il trimite la o adresa server

print(s.recvfrom(3))

s.close()

**Lab 11**

IP V4

MTU – maxim 1480

Daca vrem sa dam ping -s 30000 -> trimitem pachet de 30000 de biti trebuie sa il fragmentam in pachete de 1480 ca sa le putm trimite

Ip-urile pot fi fragmentate

La nivel de network pot fi reconstruite pachetele -> asigura ca atunci cand ajung la destinatie bucatile din pachete pot fi reconstruite

TTL – in cate noduri poate trece pachetul pana la destinatie – cand trece printr-un nod TTL-- 🡪 cand ajunge la 0 se scoate pachetul din retea

Protocol – codul protocolului la nicel networks?

Application + TCP data -> date + partea de UDP

am intrat in folderul de la lab3

git fetch origin

git reset --hard origin/master // am luat de pe git

docker-compose up -d

docker network prune

docker-compose exec client bash

root: scapy

in scapy: folosit ca sa iti creezi singur pachete

ip = IP()

ip.show()

setez ip-ul cu ip-ul serverului din docker-compose.yml

ip.dst = ‘198.13.0.14’

icmp = ICMP()

icmp.show()

icmp.type = 'echo-request' - setata asa automat

send

a,u = sr(p)

a = sr1(p)

sr(p) –> send-receive // p = pachet

sr1(p) -> send-request/receive reply one answer

p = ip/icmp

a = sr1(p)

a.show()

la IP – TTL = 63 -> era 64 dar a trecut printr-un nod

IP-ul afsat e pt ca a ajuns pachetul la destinatie -> e ip-ul reply-ului

Ip.dst = ‘198.13.0.100’

a = sr1(ip/icmp)

a.show()

Ruterul spune ca e unreacheble

Ne uitam la transport

ip = IP(dst='8.8.8.8')

udp = UDP(dport=53)

dns = DNS()

q = DNSQR()

q.show()

q = DNSQR(qname = ‘fmi.unibuc.ro’)

dns.qd = q

a = sr1(ip/udp/dns)

a.show()

Deschidem un nou terminal pe care rulam serverul

Docker-compose exec server bash

In udp-server.py -> modificam local host cu 0.0.0.0

Python elocal/laborator2/src/udp-server.py

In terminalul cu scrapy:

ip.dst = '198.13.0.14'

udp.dport = 10000

udp.sport = 12312

a = sr1(ip/udp/"salut") -> mesajul e trimis pe server

a.show()

chimbam in tcp\_server.py -> localhost cu 0.0.0.0

pe server: Python elocal/laborator2/src/tcp-server.py

pe client:

tcp = TCP(sport = 12312, dport = 10000)

tcp.seq = 20

tcp.flags='S' - putem seta flagurile ca stringuri in scapy

a = sr1(ip/tcp)

a.show()

noi am dat seq 20, el a confirmat ca a primit 21 si ne-a trimis ack 21

ack - acknowledgement

tcp.seq+=1

tcp.ack = a.seq+1

tcp.flags = ‘A’

a = sr1(ip/tcp) - nu o sa ruleze si se blocheaza

exit() - din scapy

in lab 3 in tcp\_handshake.py am modificat:

am comentat ip.src=

ip.dst = ‘198.13.0.14’

si la final rcv.show() – penultima linie

in clientul din docker rulam python elocal/laborator3/src/tcp\_handshake.py