SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 479/980: Systems and Network Security
Spring 2020

Network Analysis – Part 2

Instructor: Khaled Diab

Goal

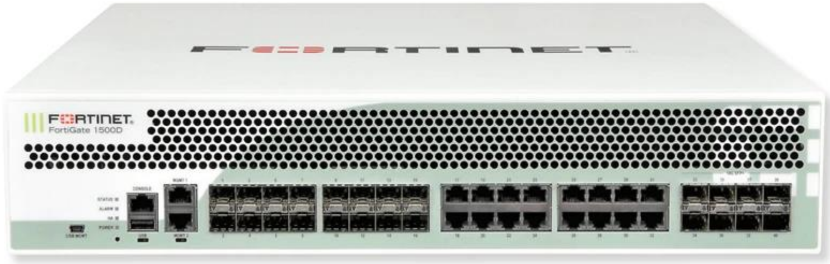
- Analyze network traffic for different goals.

- Useful for:
  - Intrusion Analyst: dissect network traffic to study intrusions
  - Forensic Investigator: check the extent of a malware infection
  - Attackers: understand their victim networks!

Outline

- Network Hardware
- Packets
  - Dissecting Packets
  - Sample of Network Protocols
    - ARP and ICMP
- Network Security Monitoring
  - Data Collection
  - Packet filtering
  - Tools: Wireshare
- Network-level operations:
  - Network Reconnaissance
  - Traffic Manipulation
    - Spoofing

Phases of Network Security Monitoring



Network

Collection → Detection → Analysis

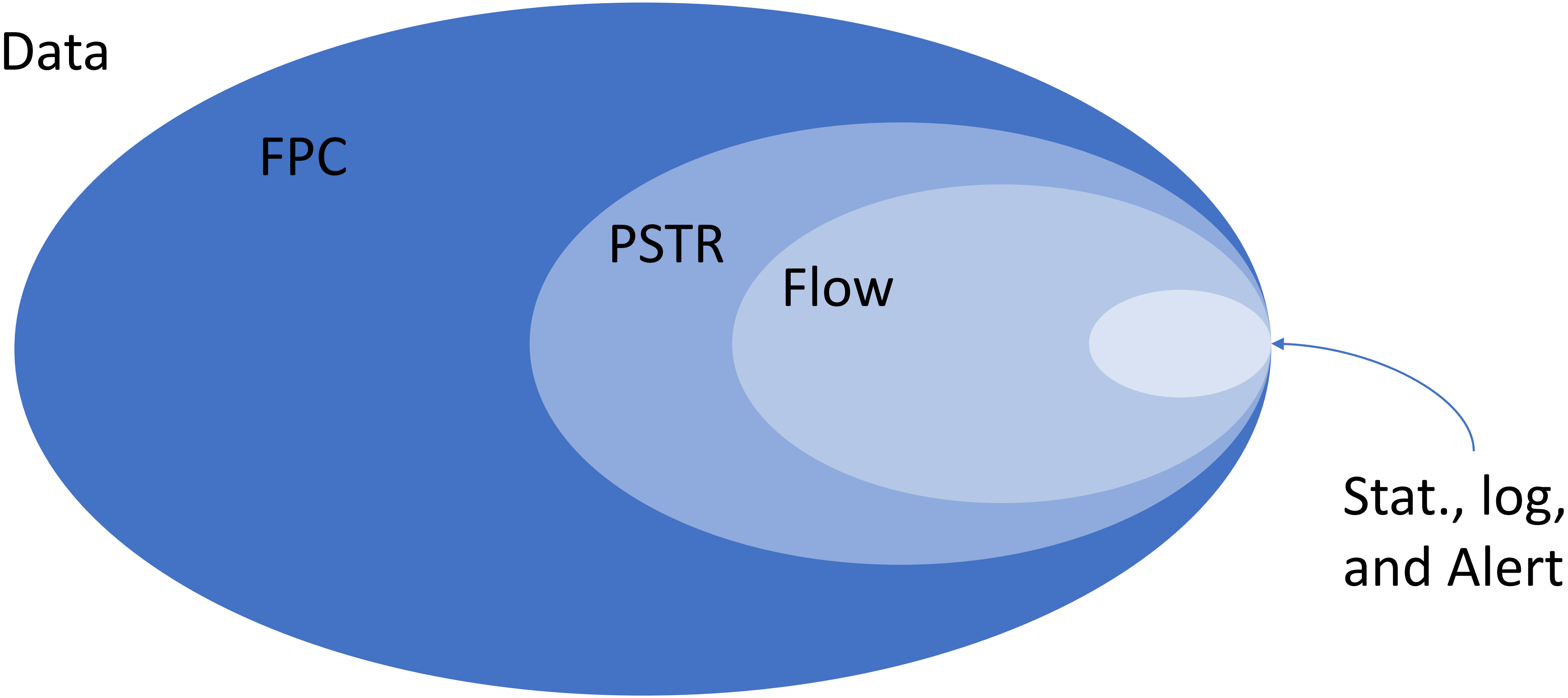
Some devices can perform the three operations

Data Types

- Full Packet Capture (FPC) Data
  - All transmitted packets
  - Popular format is `pcap`
  - Large size but useful for analysis
  - Used to derive other data types
- Session Data
  - Summary of communication between two devices
  - Aka a *flow* or *conversation*
  - Small size → can be retained for longer time
- Packet String (PSTR) Data
  - Intermediate data between FPC and Session data
  - Clear test strings from a protocol header (e.g., HTTP)
  - Closer granularity to FPC but smaller size

Data Types

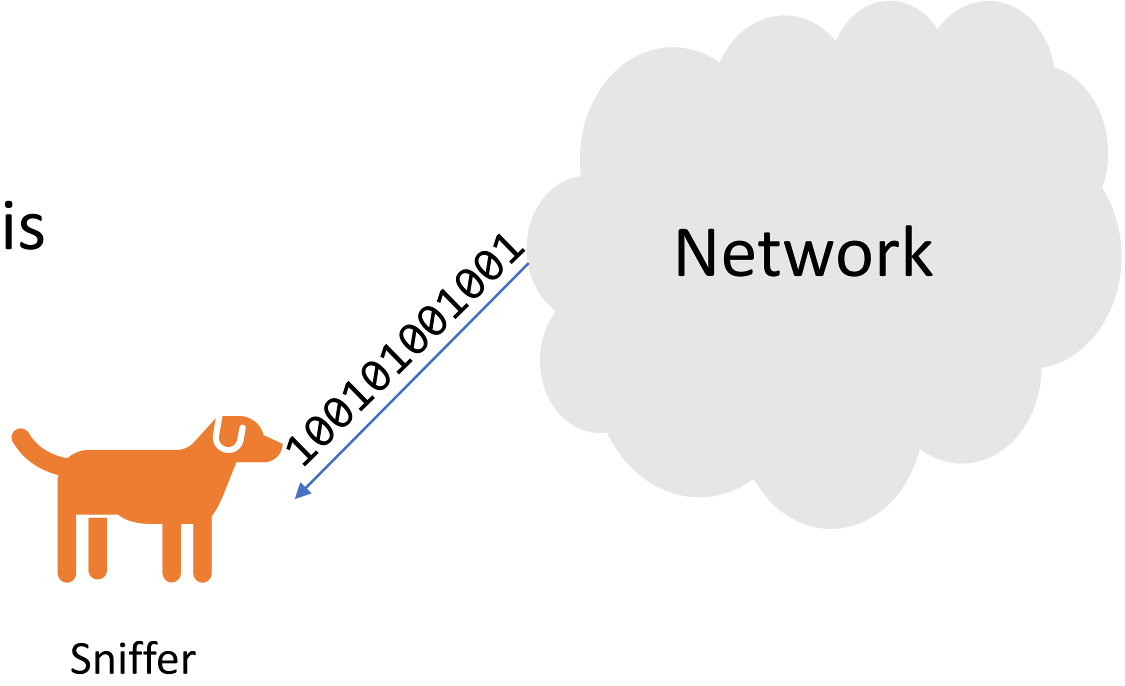
- Statistical Data
- Log Data
- Alert Data



Data Collection

Sniffing Packets

- The process of capturing network traffic (i.e., packets)
  - By a sniffer (or a sensor)
- Packets are stored for further analysis
- This requires modifications to:
  - The network
  - The sniffer



Tapping into the Wire

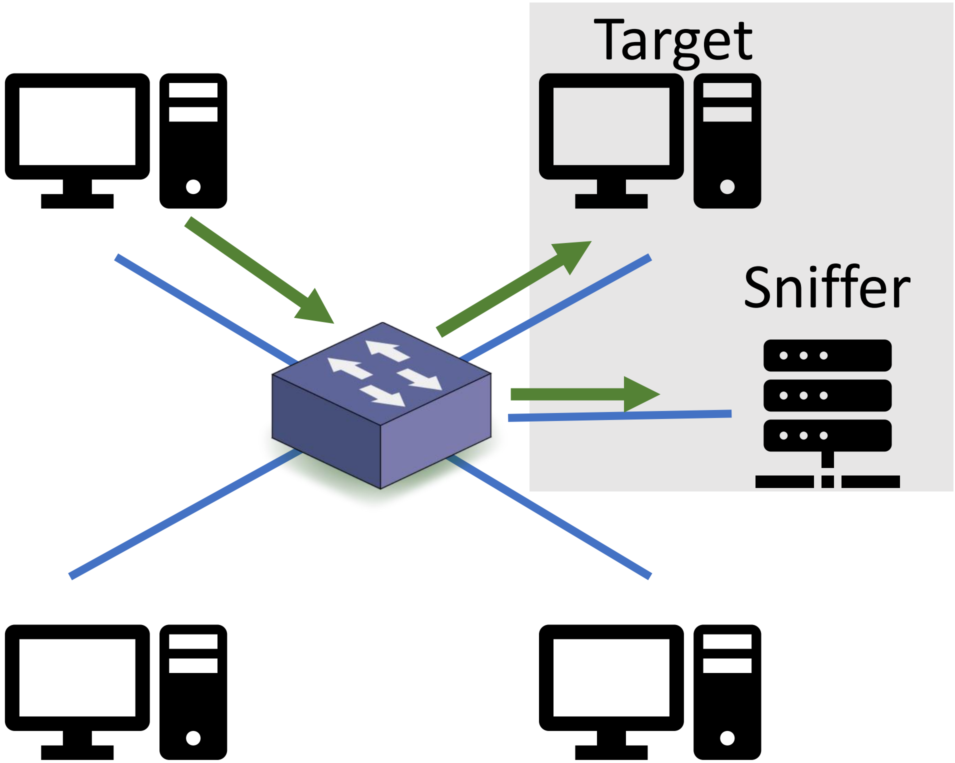
- How can a sniffer capture traffic?

- Three techniques in switched networks:
  - Port mirroring
  - Installing a Hub
  - Network TAP

Port Mirroring

- Copies traffic from one port to another
- Easy way to capture traffic
- Low-cost option

- Requirements:
  - Access to switch command line
  - Support of port mirroring
  - Available port

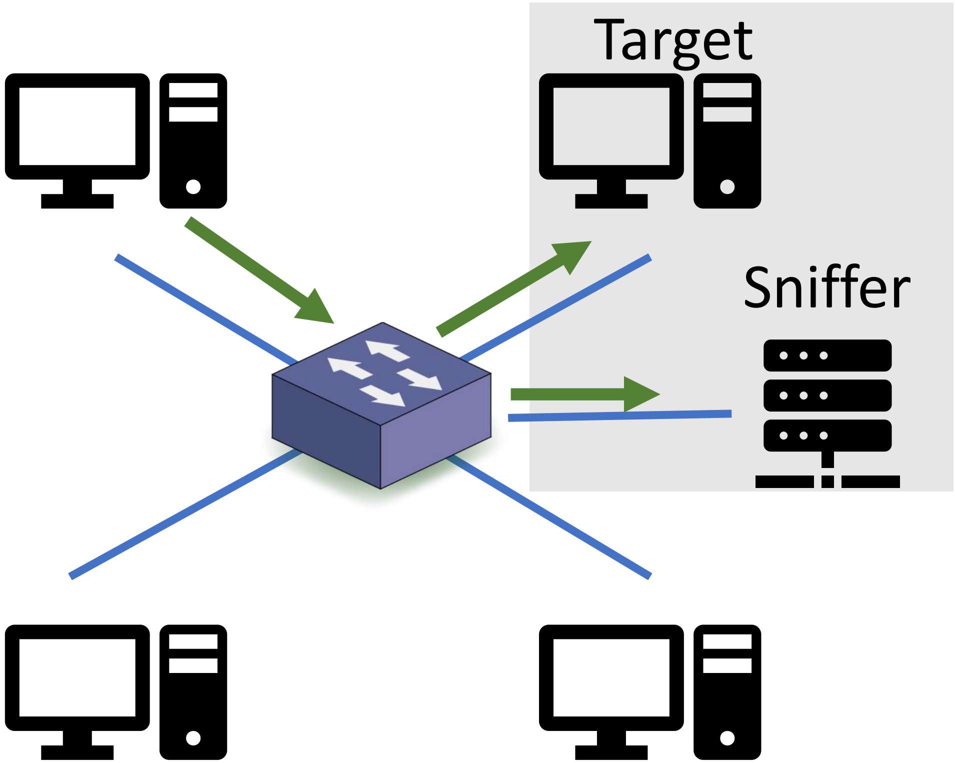


Port Mirroring: Configuration

- Configuring port mirroring on a switch is vendor-specific.
  - Usually happens through command line
  - Sometimes through GUI or web interface

- For example, for Cisco switches:

```
set span <src_port> <dst_port>
```

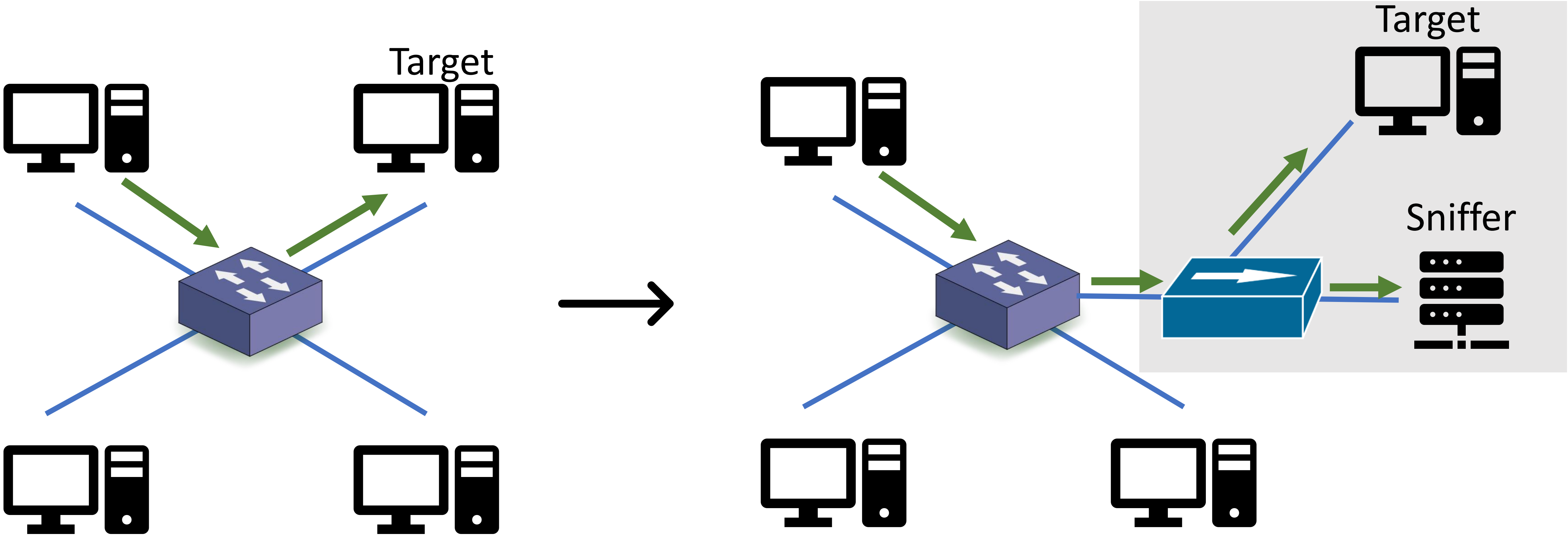


Port Mirroring

- In general, port mirroring is not reliable for high-throughput applications:
  - Example: network security monitoring

- Why?
  - If multiple ports are mirrored to a single port (oversubscription)
    - Packet losses
    - Slowing down the switch

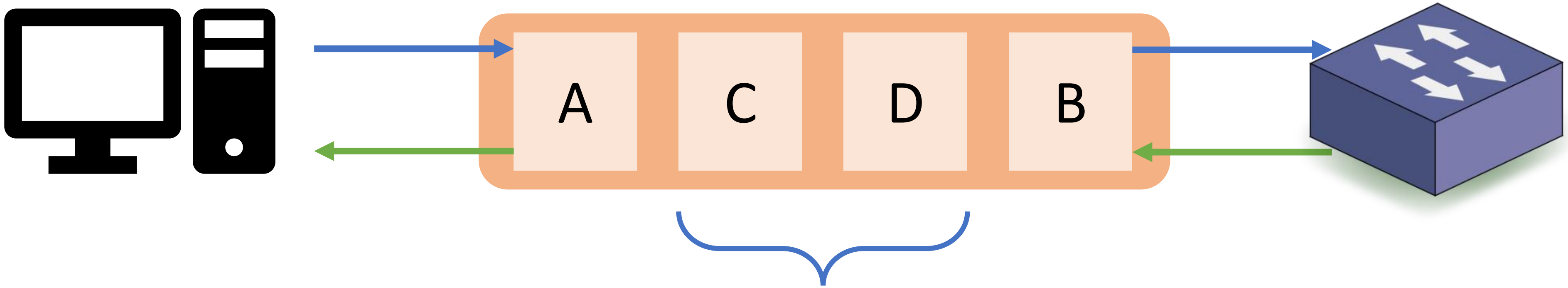
Installing a Hub



Issues?

Installing a TAP

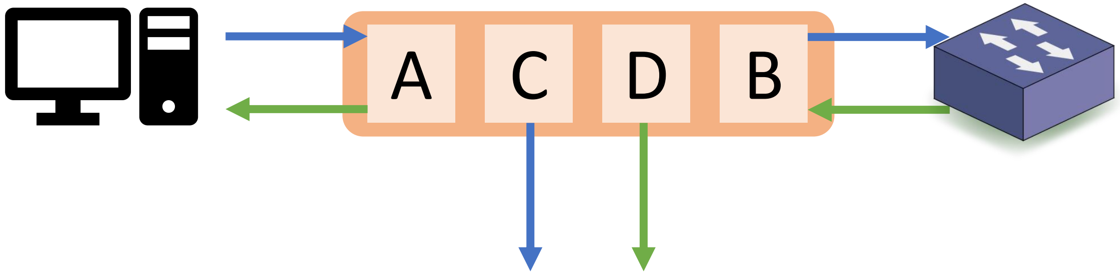
- TAP: Test Access Point
- Specialized hardware that allows traffic to flow:
  - from port A → port B, and
  - from port B → port A
- Creates an exact copy of both sides of the flow
  - Without loss



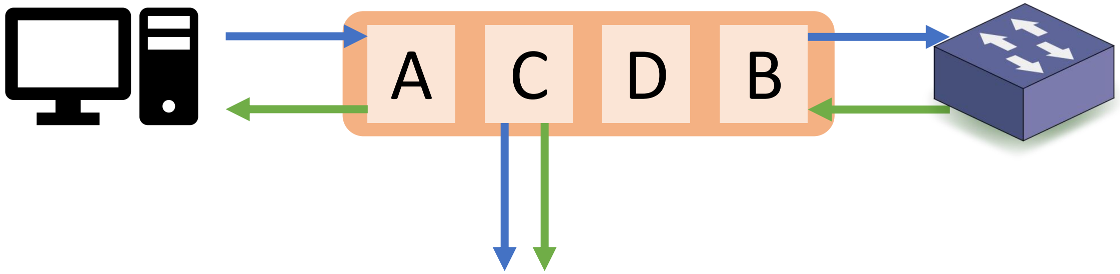
To a sniffer/monitoring device

Installing a TAP: Modes

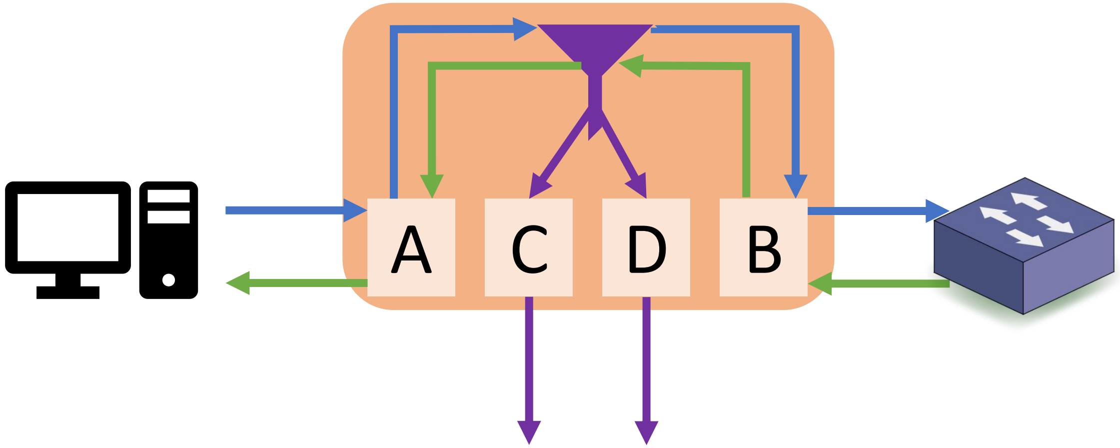
Breakout Mode



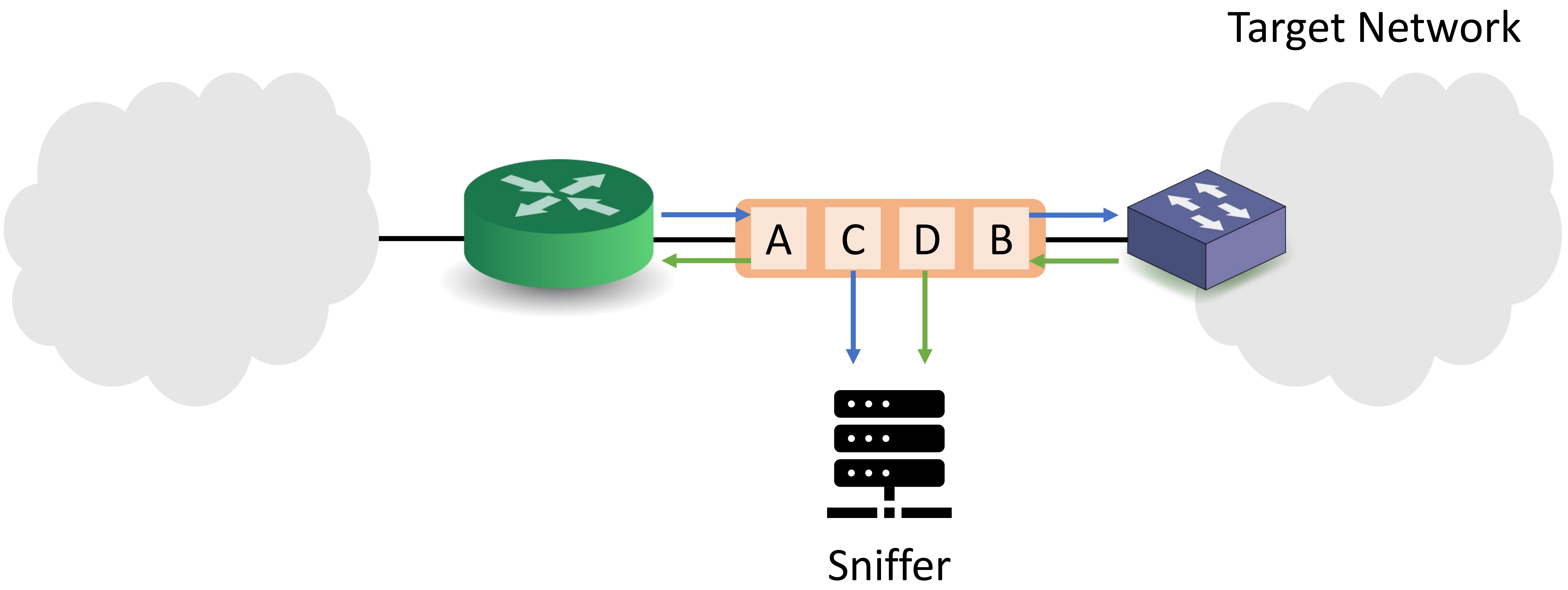
Aggregation Mode



Filter Mode

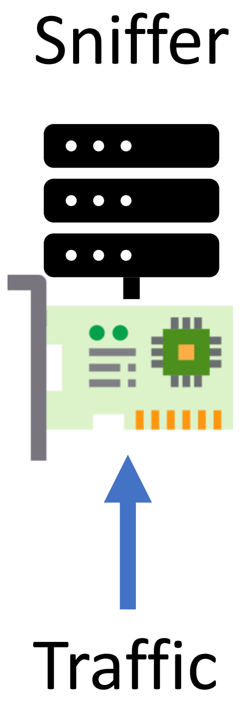


Sniffing Network Segment



Sniffer Machine

- The sniffer receives network traffic
  - that wasn't destined for the sniffer
- This happens in some network protocols as well.
  - Examples?
- The default behavior of NIC is to discard these packets!
  - Reduce CPU processing
  - Not useful for the sniffer!
- How can we solve this issue?



NIC: Promiscuous Mode

- Recent NICs support “promiscuous” mode
  - Allows the NIC to receive traffic not destined for the sniffer
  - The NIC then passes sniffed packets to the CPU for further processing

- Check an interface:

loopback intf

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1
...
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP group default qlen 1000
```

NIC intf

NIC: Promiscuous Mode

- Enable promiscuous mode:

```
$ sudo ip link set enp0s3 promisc on
```

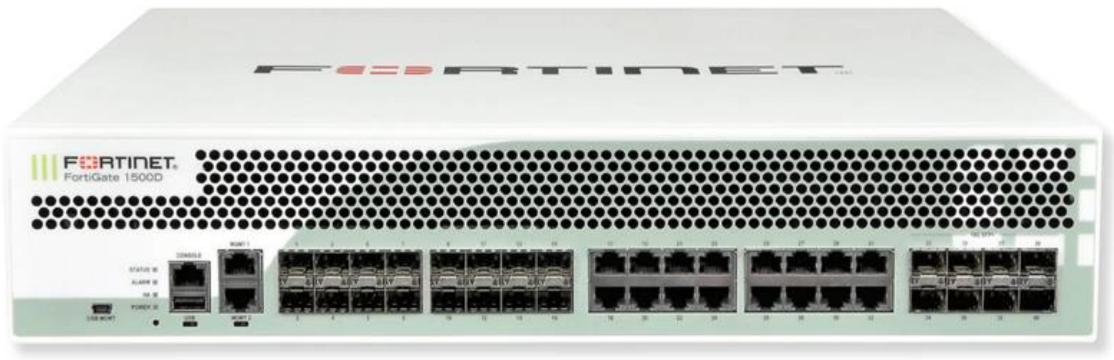
- Check again:

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1
...
2: enp0s3: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500
qdisc pfifo_fast state UP group default qlen 1000
```

PROMISC enabled

Sniffer Machine: Other Aspects

- CPU
- Memory
- Storage
- OS
- Cabling
- Cooling
- …



Packet Filtering

- Capture or show packets matching specific fields or criteria
- Packet filtering is used during:
  - The capturing phase. Sniffer may eliminate:
    - unwanted traffic, or
    - traffic that isn’t useful for detection/analysis
  - The analysis phase:
    - Analysts often need to focus on specific packets
      - E.g., HTTP packets, ARP requests, Ping (echo reply), etc.
- Berkeley Packet Filter (BPF) is the most commonly used syntax

Berkeley Packet Filters (BPFs)

- McCanne and Jacobson'93 https://www.tcpdump.org/papers/bpf-usenix93.pdf:
  - Filters are translated into a simple instruction/register set used to specify if packets are to be rejected, accepted
  - A simple VM ran the instructions in-kernel and filtered appropriately
  - Safety was the key criterion when injecting filter code.
    - All programs must complete in a bounded time (no loops)

BPF Syntax

$$\text{Expression} = \overbrace{\underbrace{\texttt{udp}}_{\text{Qualifier}}\ \underbrace{\texttt{port}}_{\text{Qualifier}}\ \underbrace{\texttt{53}}_{\text{Value}}}^{\text{Primitive}}\ \underbrace{\texttt{\&\&}}_{\text{Operator}}\ \overbrace{\texttt{dst host 192.0.2.2}}^{\text{Primitive}}$$

BPF Syntax

- Three types of qualifiers:
  - **type**: `host, net, port, portrange`
  - **dir**: `src, dst`
  - **proto**: `ether, arp, ip, ip6, icmp, tcp, udp`

```
tcp port 80
```

```
ip host 10.0.0.1
```

=

```
ether proto \ip and host 10.0.0.1
```

BPF Syntax

- Match specific fields in the packet:
  - `icmp[0] == 8`

Internet Control Message Protocol (ICMP)					
Offsets	Octet	0	1	2	3
Octet	Bit	0–7	8–15	16–23	24–31
0	0	Type	Code	Checksum	
4+	32+	Variable			

`0` : Echo Reply
`8` : Echo Request
`11`: Time Exceeded

BPF Syntax

- Match specific fields in the packet:
  - `ip[8] > 64`

Internet Protocol Version 4 (IPv4)							
Offsets	Octet	0		1	2		3
Octet	Bit	0–3	4–7	8–15	16–18	19–23	24–31
0	0	Version	Header Length	Type of Service	Total Length		
4	32	Identification			Flags	Fragment Offset	
8	64	Time to Live		Protocol	Header Checksum		
12	96	Source IP Address					
16	128	Destination IP Address					
20	160	Options					
24+	192+	Data					

BPF Syntax

- Match specific fields in the packet:
  - `tcp[14:2] == 0`

Transmission Control Protocol (TCP)						
Offsets	Octet	0		1	2	3
Octet	Bit	0–3	4–7	8–15	16–23	24–31
0	0	Source Port			Destination Port	
4	32	Sequence Number				
8	64	Acknowledgment Number				
12	96	Data Offset	Reserved	Flags	Window Size	
16	128	Checksum			Urgent Pointer	
20+	160+	Options				

APIs and Tools

- Scapy
- `libpcap`
- `tcpdump`
- `nmap`
- Wireshark
- `tshark`
- …

Wireshark

- Some features:
  - Packet Filters: Display and Capture (BPF)
    - Wiki: https://wiki.wireshark.org/CaptureFilters
  - Statistics
  - Follow Stream

Wireshark

Demo

Network Reconnaissance

Network Reconnaissance

- Goal: Perform in-depth research on the target system

- Two techniques:
  - Port scanning
  - OS fingerprinting

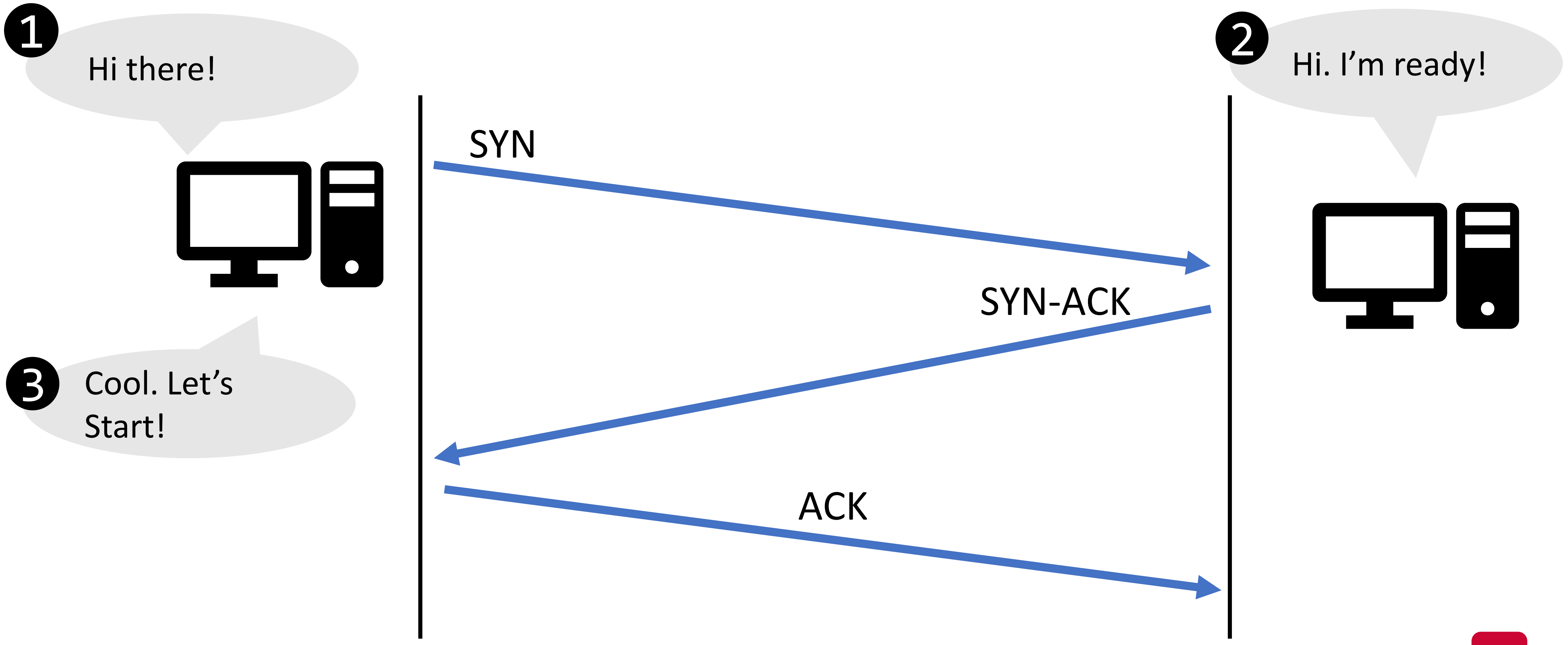
Port Scanning

- Goals:
  - to determine whether the victim is alive and reachable
  - to know which ports the victim is listening to

- TCP SYN scan
  - Fast and reliable
  - Portable across platforms
  - Less noisy than other techniques

TCP: Connection Establishment

- Any TCP connection starts with a three-way handshake.



TCP SYN Scan

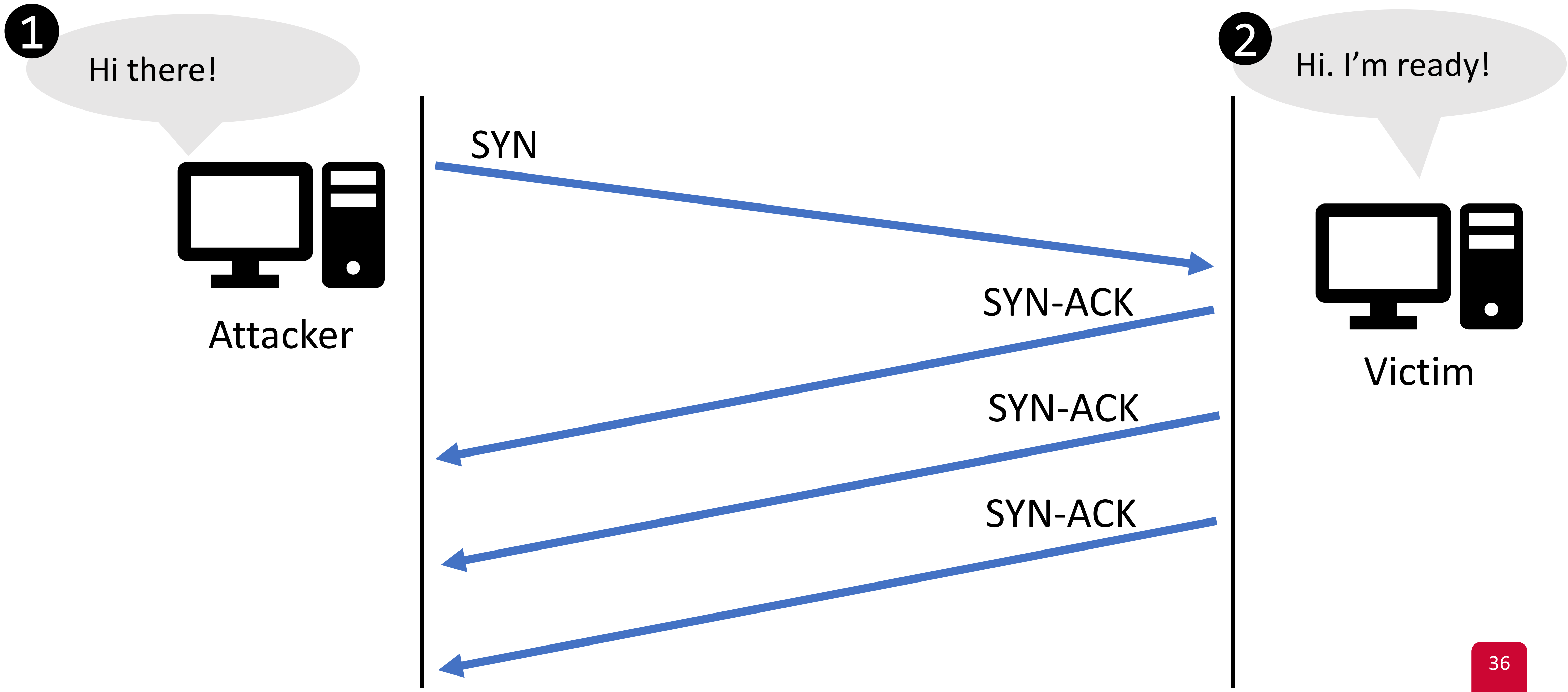
- SYN scan relies on the three-way handshake in TCP.
  - Using *half-open* connection! Other consequences?

- The attacker determines a port is open based on:
  - the packet sent by the victim (if any)

- Three possible cases.

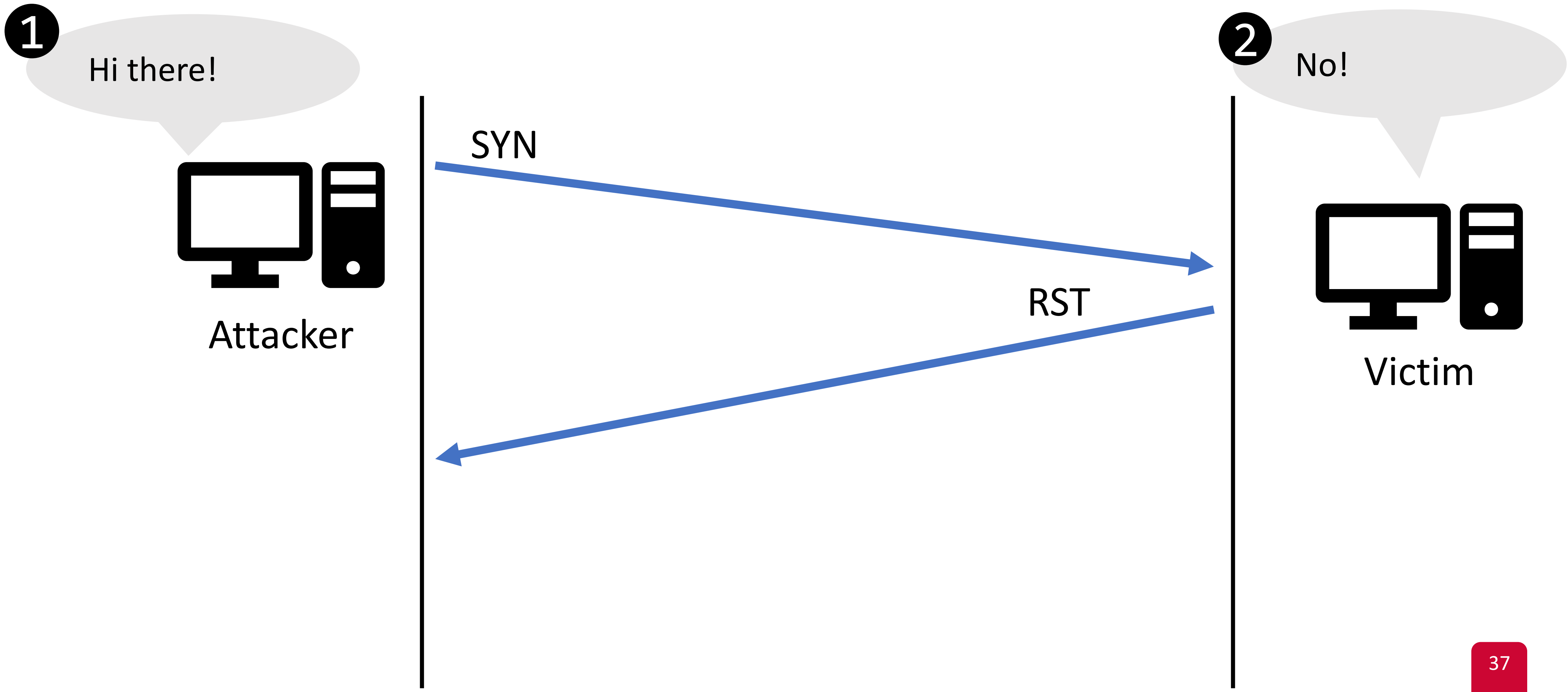
TCP SYN Scan: Case 1

- The victim replies with SYN-ACK → The attacker knows that the port is open.



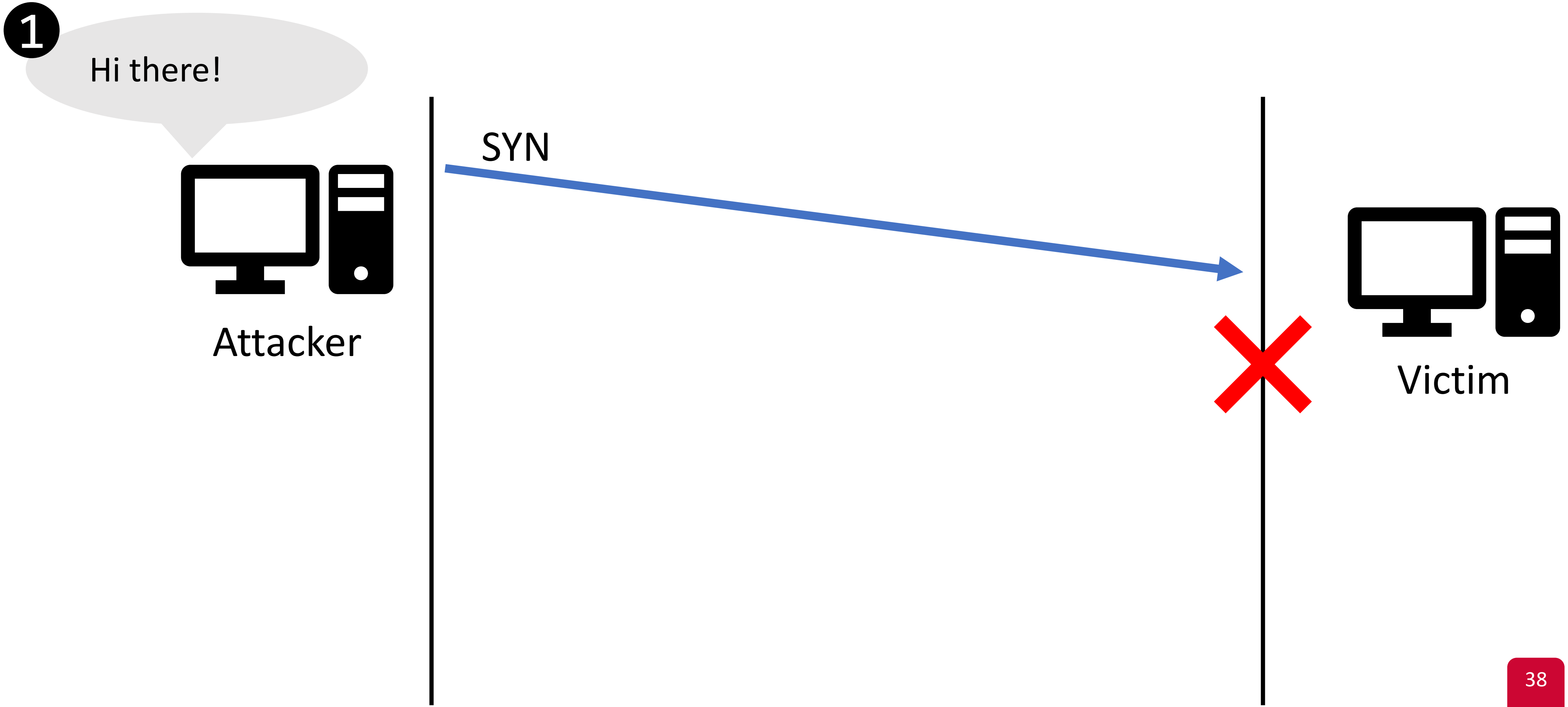
TCP SYN Scan: Case 2

- The victim replies with RST → The attacker knows that the port is closed.



TCP SYN Scan: Case 3

- The attacker does not receive a response → inconclusive.



Analyzing SYN Scan in Wireshark

- An example formed by using `nmap`
- We will use the Conversation window to check TCP handshake

- Conversations having:
  - 5 pkts → indicates that the port is open
  - 2 pkts → indicates that the port is closed
  - 1 pkt → inconclusive!

OS Fingerprinting

- Determining the victim's OS without having physical access to the machine.

- Useful to:
  - configure the methods of attack
  - know the location of critical files
  - E.g., some versions of OSs have certain vulnerabilities

Passive OS Fingerprinting

- Examine certain fields within packets to determine the OS
- The attacker needs only to listen to packets
  - And does not need to send any packet!
  - Ideal because the attacker is stealthy

- Key Idea:
  - Standards tell us the fields belonging to a protocol
  - But, they don't tell us the default values of many fields!
  - Many of these default values are OS-specific

Common Default Values – IP

Field	Default Value	Platform
Initial TTL	64	nmap, BSD, OS X, Linux
	128	Windows
	255	Cisco IOS, Solaris
Don't Fragment flag	Set	BSD, OS X, Linux Windows, Solaris
	Not set	Nmap, Cisco IOS

Common Default Values – TCP

Field	Default Value	Platform
Window Size	1024—4096	nmap
	65535	BSD, OS X
	Variable	Linux, Windows
	4128	Cisco IOS
	24820	Solaris
Max. Segment Size	0	nmap
	1440—1460	Windows
	1460	BSD, OS X, Linux, Solaris
SackOK	Set	Linux, Windows, OS X
	Not set	nmap, Cisco IOS, Solaris

Passive OS Fingerprinting

- Demo using Wireshark

- Open source tools:
  - `p0f`: http://lcamtuf.coredump.cx/p0f3/

Traffic Manipulation

Traffic Manipulation

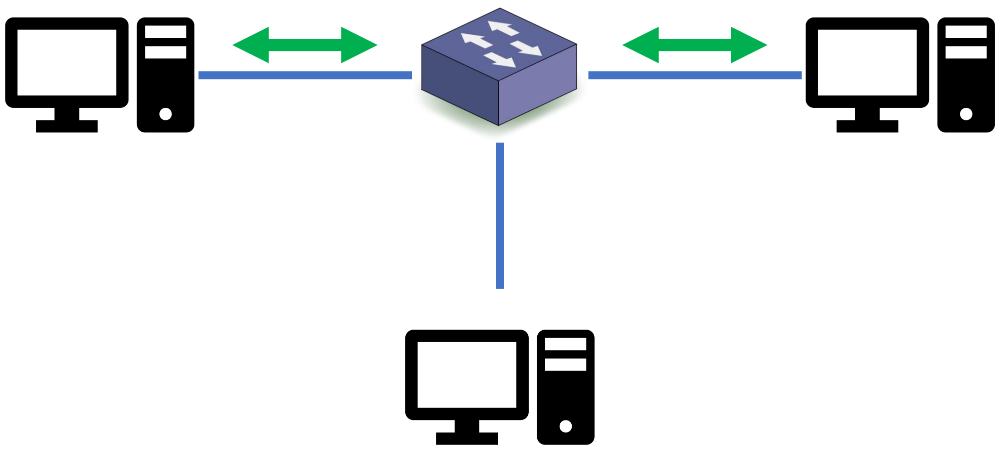
- This is done by means of packet spoofing:
  - Pretend to be someone else by creating a packet with specific values

- Results in a man-in-the-middle (MITM) attack.
- An attacker redirects traffic between two hosts
  - To intercept or modify data in transit

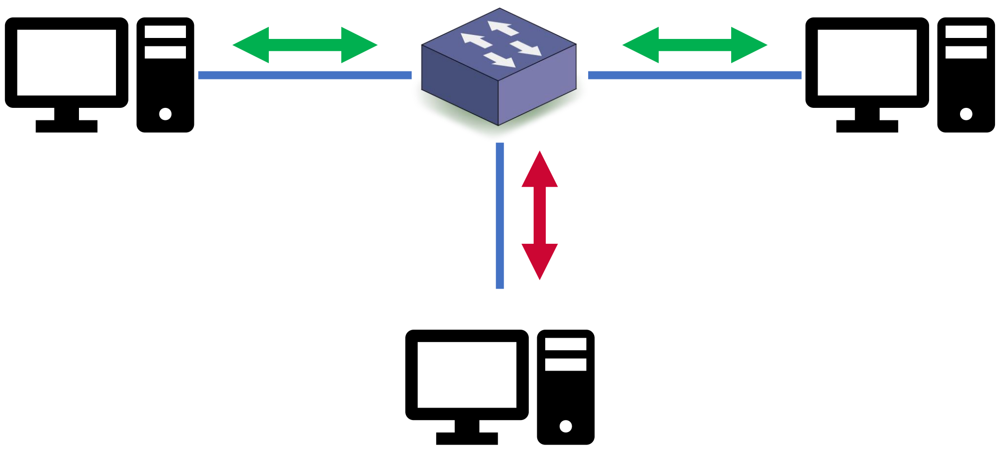
ARP Cache Poisoning

- A crafted ARP packet:
  - tricks two endpoints into thinking they're communicating with each other
  - but, they are communicating with the attacker!
- Consequences: DoS, MITM (e.g., HTTP session hijacking).

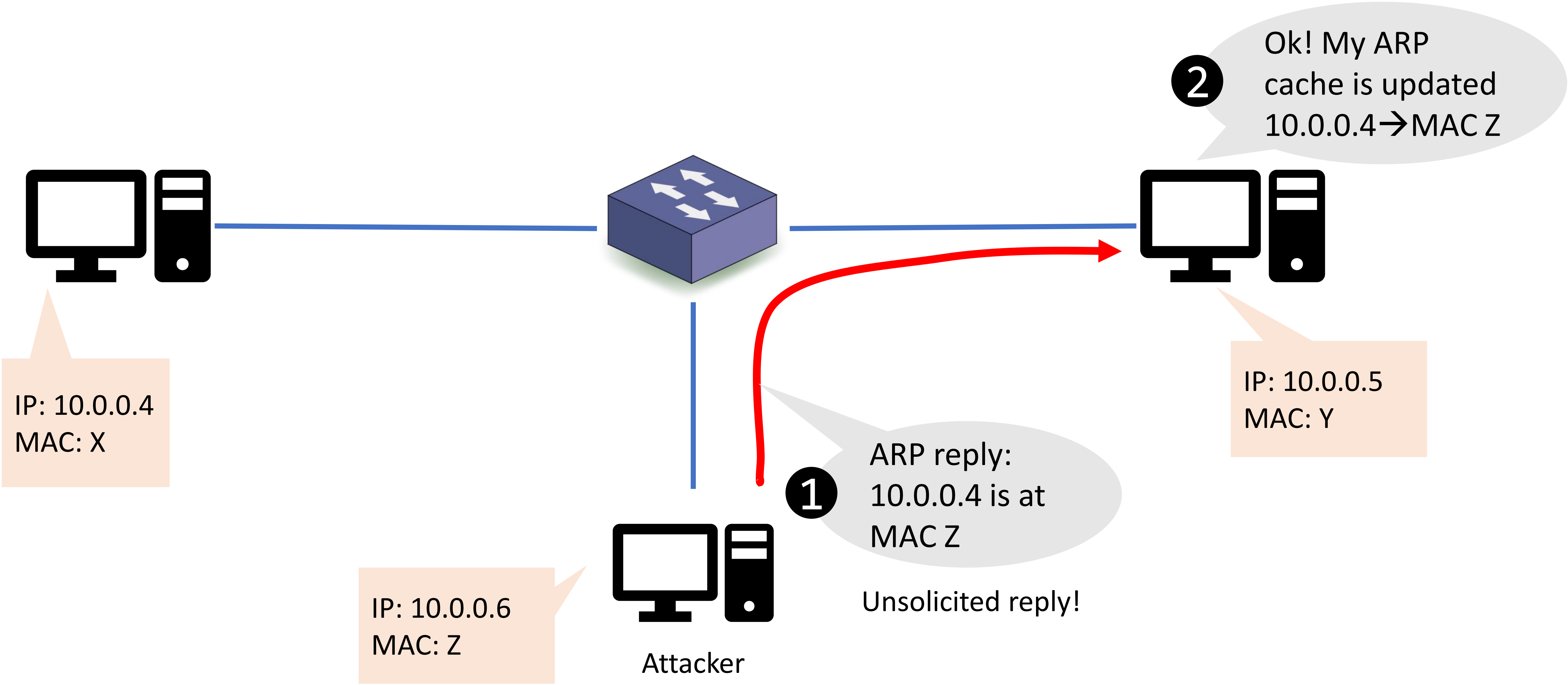
Regular Scenario



ARP Cache is Poisoned



ARP Cache Poisoning



ARP Cache Poisoning: Root Cause

- ARP is a stateless protocol
- ARP hosts don’t authenticate ARP replies:
  - Even if a host doesn’t send an ARP request.
  - Overwrites an ARP entry (even if it hasn’t expired)!

ARP Cache Poisoning: Defenses

- Static ARP entries:
  - Cannot be changed by the attacker
  - Good for small networks (or networks that don’t change)

- IDS or Ethernet switches
  - Detect unsolicited replies.

Summary

- Network security monitoring
- Packet sniffing and spoofing
- ARP cache poisoning
- How to implement (parts of) many tools:
  - `ping`, `traceroute`, `nmap`, `p0f`, Cain & Abel

To do list

- Start using Wireshark
- Get familiar with packet diagrams and major protocols:
  - IP, ARP, ICMP, DNS, TCP, UDP
- Summarize [R12]
- In three weeks: Project milestone presentation

Next Lecture

- TCP/IP Attacks