Lecture 6
Firewall and NIDS

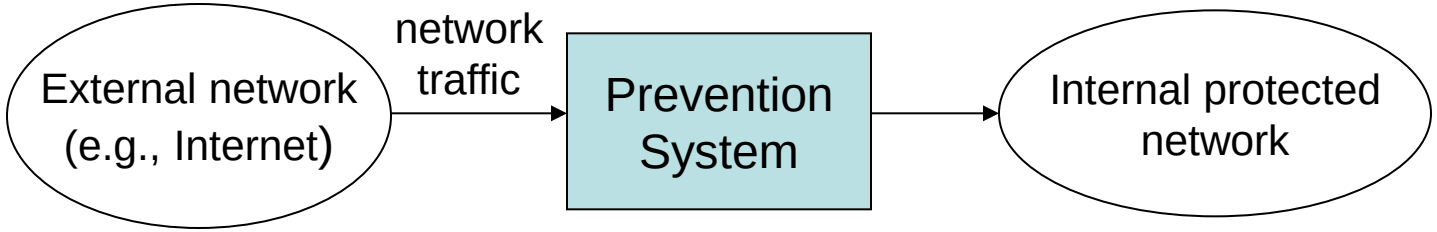
ENGG5105/CSCI5470 Computer and Network Security

Spring 2014

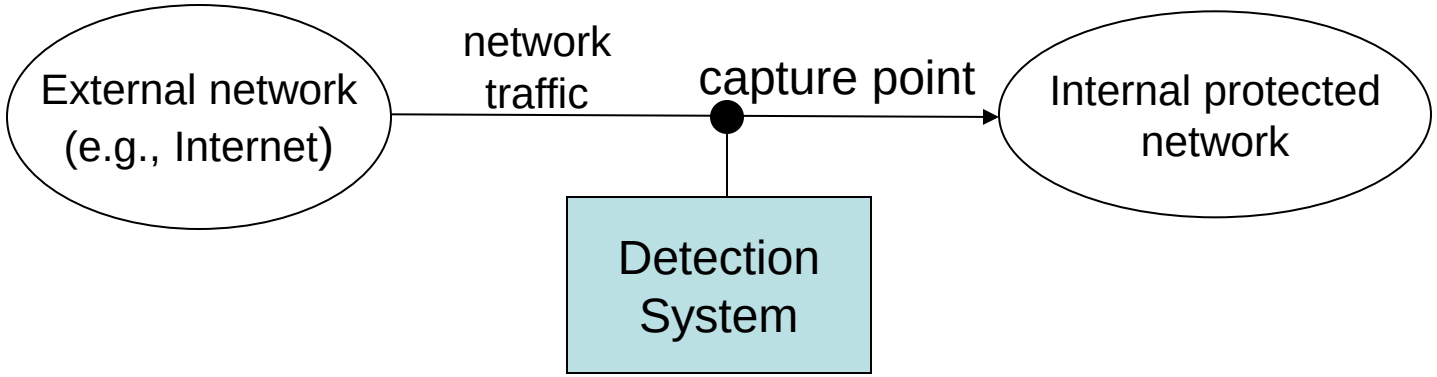
Patrick P. C. Lee

How to Defend Against Network Attacks?

- **Prevention**: block/deny attacks
  - e.g., firewall



- **Detection**: detect attacks, no immediate reaction
  - e.g., NIDS (network intrusion detection system)



Roadmap

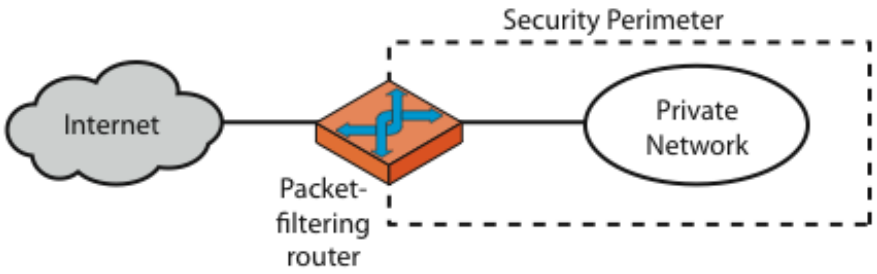
- Firewall
- NIDS (Network Intrusion Detection System)

What is a Firewall

- A **choke point** of control and monitoring
- Interconnects networks with differing trust
- Imposes restrictions on network services
  - only authorized traffic is allowed
- Auditing and controlling access
  - can implement alarms for abnormal behavior
- Must be immune to penetration

Deploying Firewalls

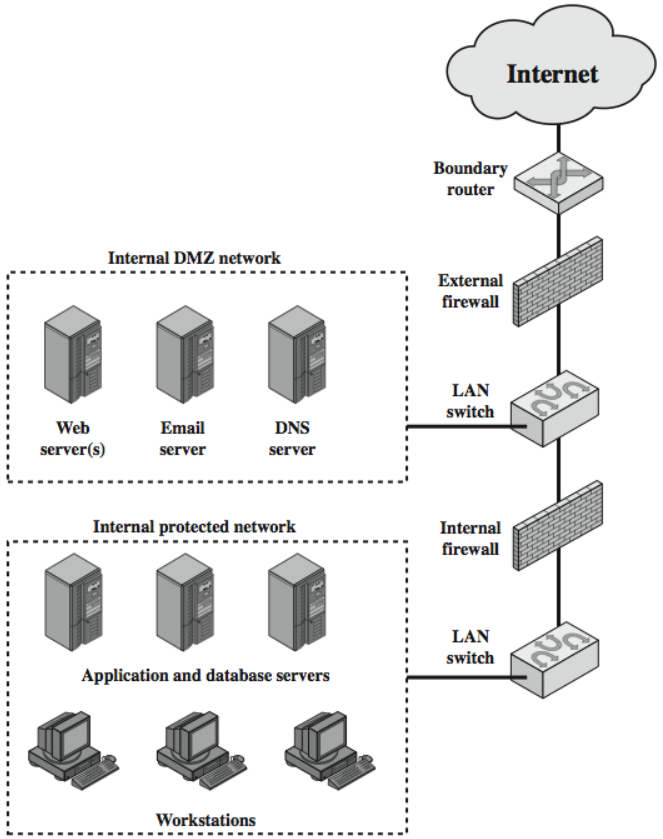
- We can deploy a firewall that interconnects external network (untrusted) and internal network (trusted)



(a) Packet-filtering router

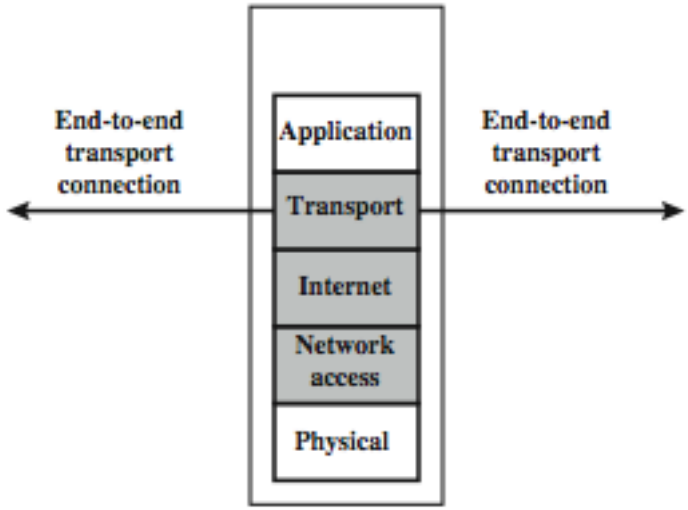
Deploying Firewalls

- Deploy multiple firewalls with different policies
- Internal firewalls have more stringent policies to protect internal hosts
- A DMZ (demilitarized zone) between the external and internal firewalls to deploy public services



Firewall – Packet Filters

- Simplest, fastest firewall component
- Foundation of any firewall system
- Examine each IP packet and permit or deny according to rules
  - e.g., you can define a rule that blocks any inbound packets to port 53 (DNS service)
  - hence restrict access to services (ports)



Firewalls – Packet Filters

Table 20.1 Packet-Filtering Examples

A

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

B

action	ourhost	port	theirhost	port	comment
block	*	*	*	*	default

C

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connection to their SMTP port

D

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		our packets to their SMTP port
allow	*	25	*	*	ACK	their replies

E

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

Firewall – Packet Filters

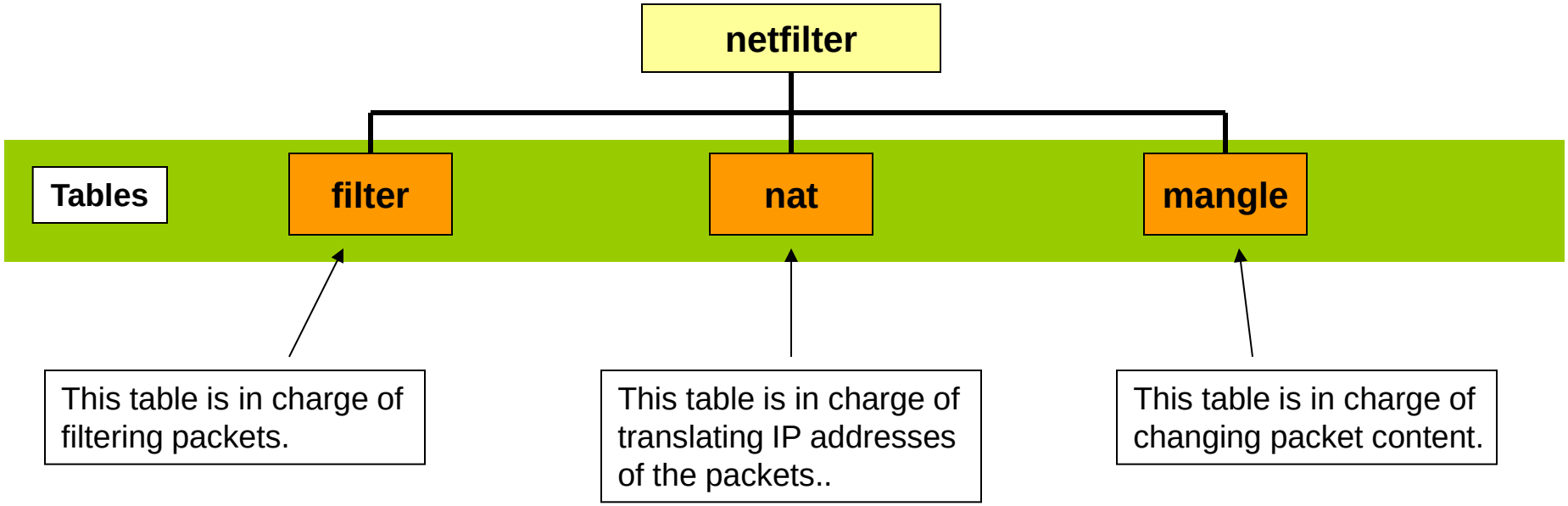
- Can implement stateful filtering
- Stateful packet filters examine each IP packet in context
  - keep track of client-server sessions
  - check each packet validly belongs to one
- Add more intelligence, but with higher processing overhead

iptables

- iptables is a user-level Linux program that controls the kernel-level network module called netfilter. It can perform:
  - Packet Filtering
  - Packet Forwarding
  - Network Address Translation (NAT)
  - Connection Tracking (stateful tracking)
- We can use iptables to implement a filter-based firewall atop Linux.

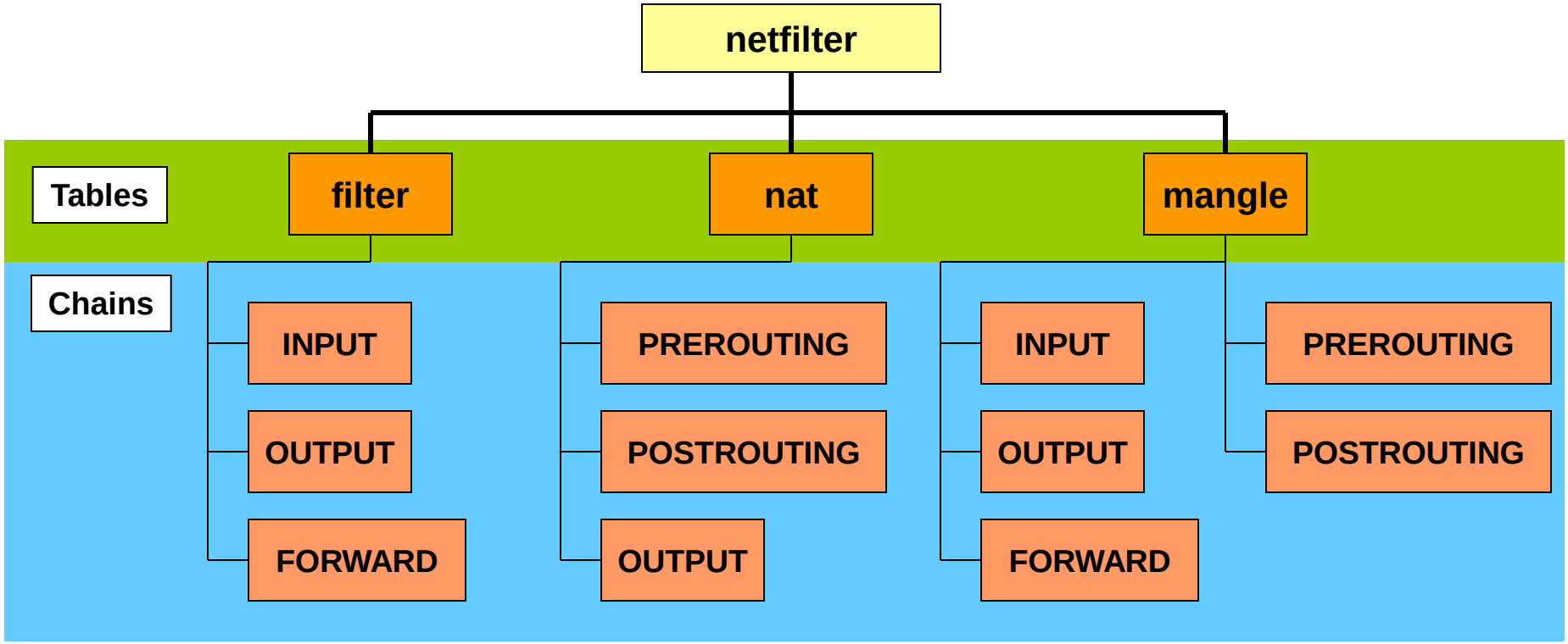
iptables – Tables and Chains

- Each function provided by the netfilter architecture is presented as a **table**.

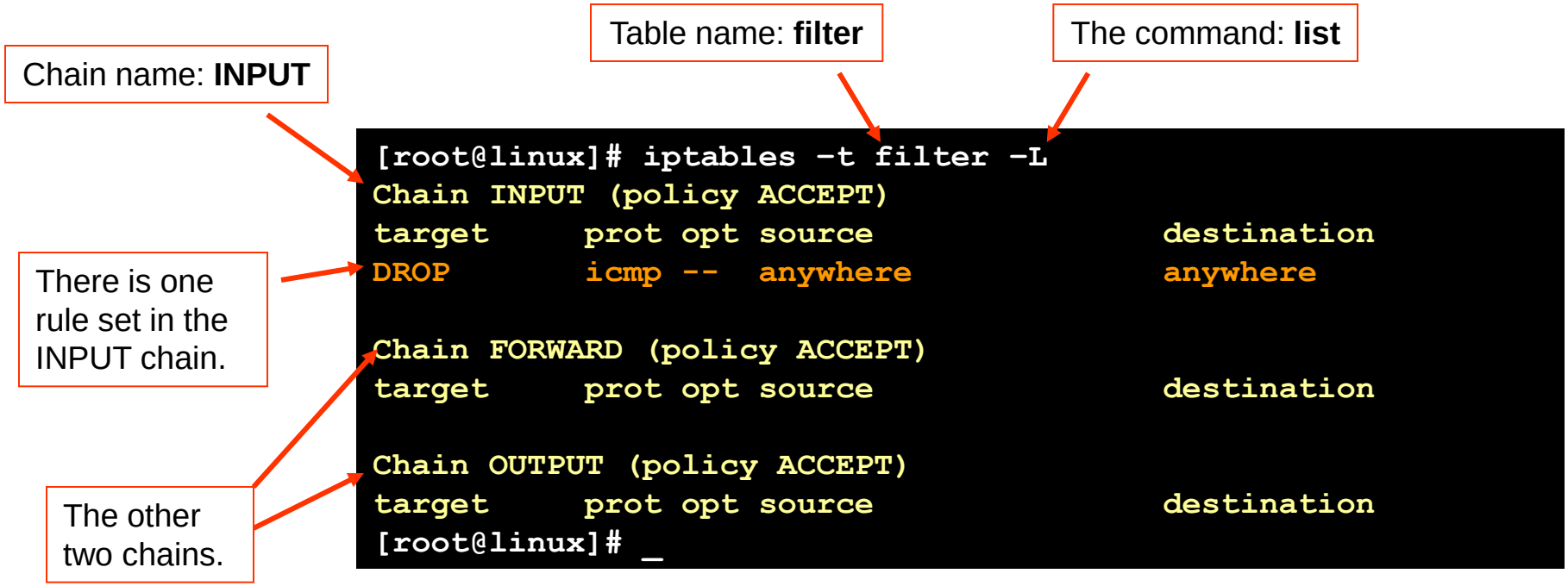


iptables – Tables and Chains

- Under each table, there are a set of **chains**.
  - Under each chain, you can assign a set of **rules**.



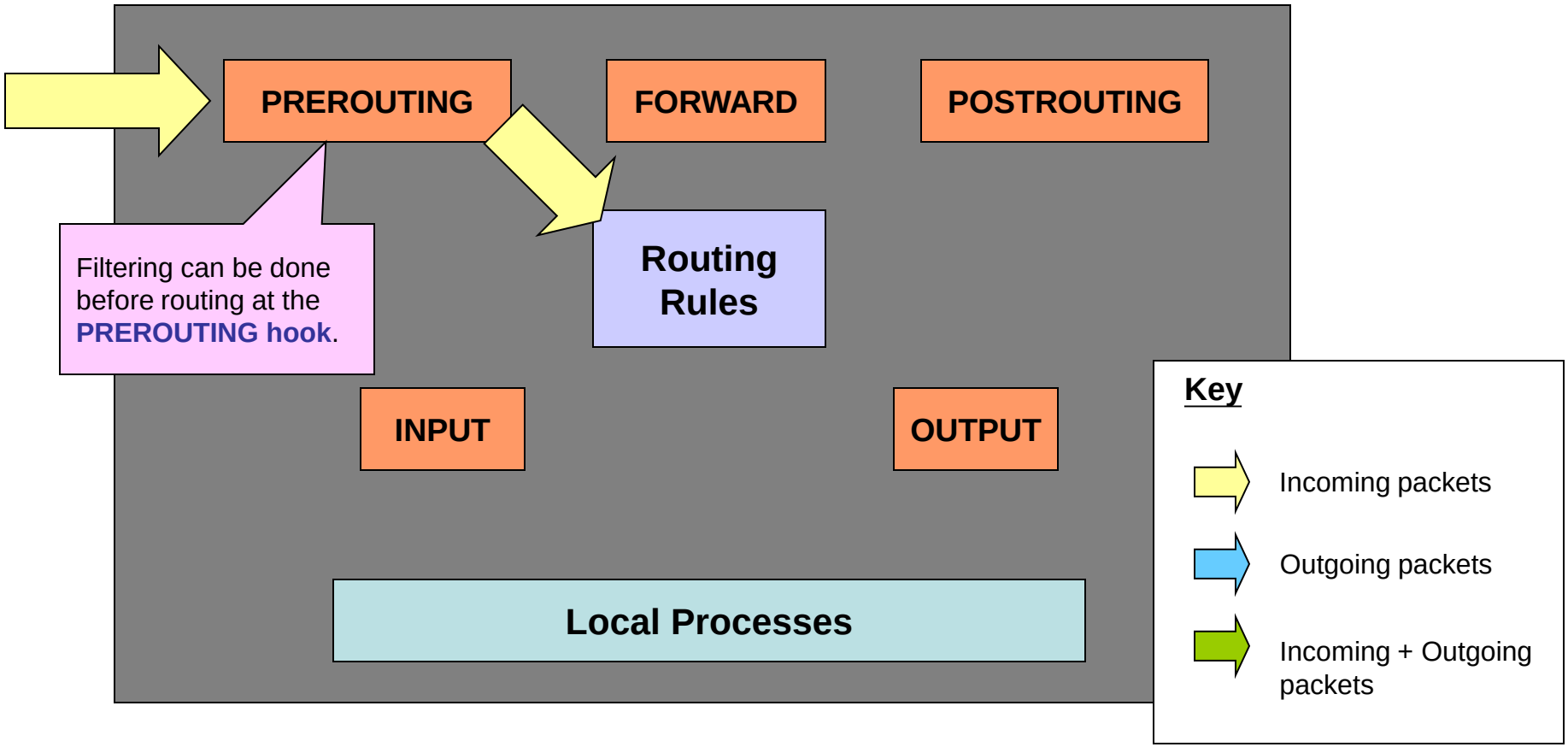
iptables – Tables and Chains



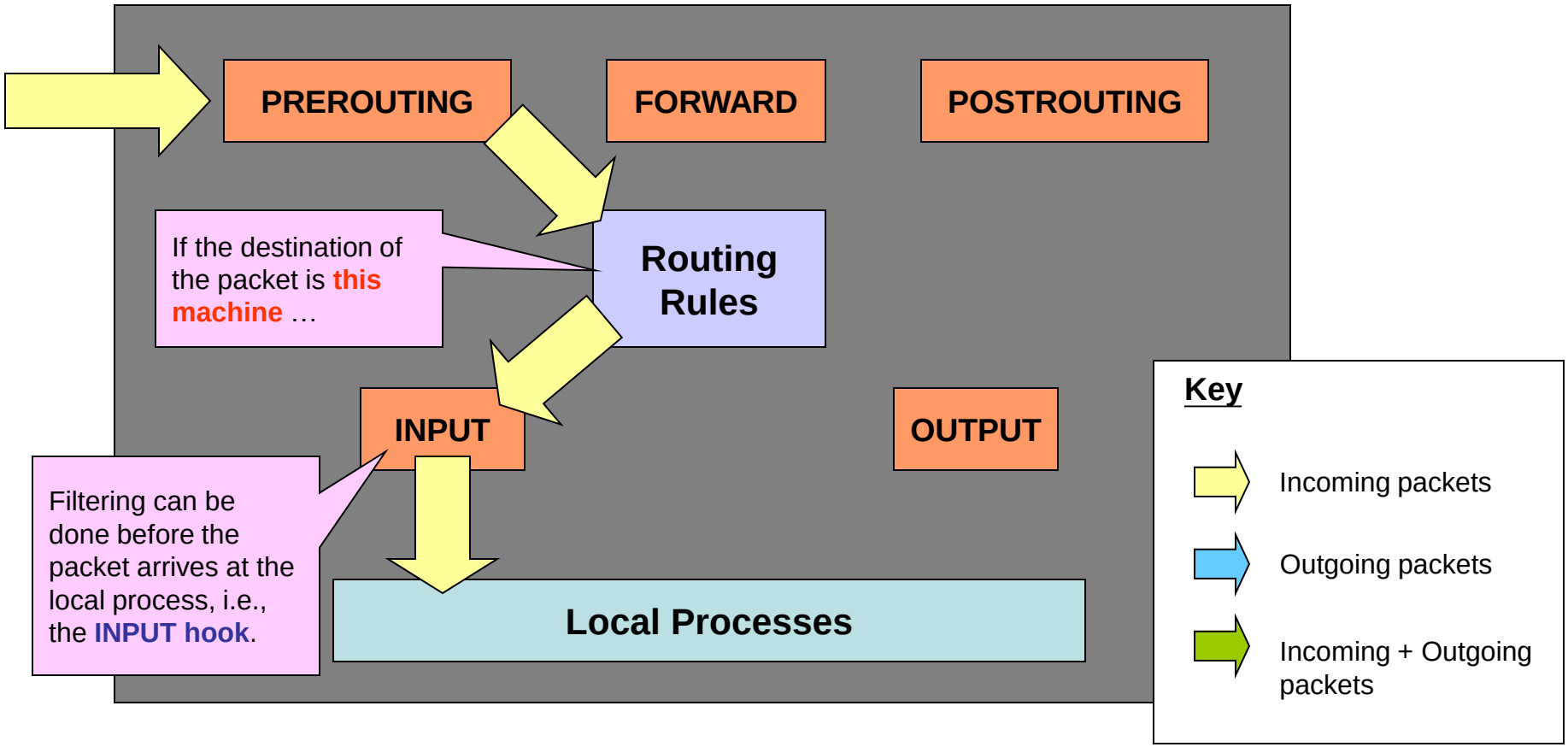
The rule in the INPUT chain means:

When a packet with ICMP payload passes through the **INPUT hook**, **DROP** that packets, no matter it is **from anywhere** and **to anywhere**.

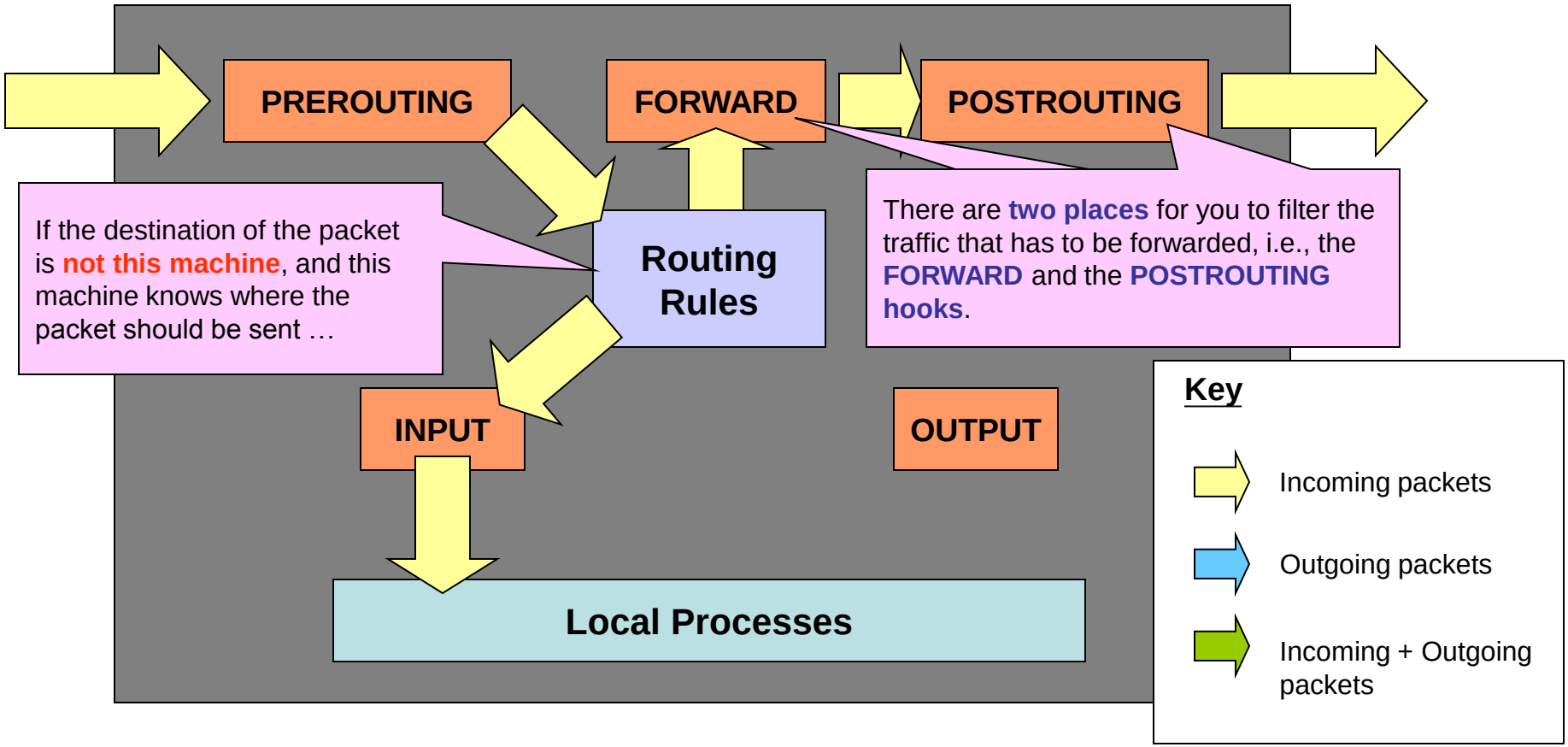
iptables – Packet Flow



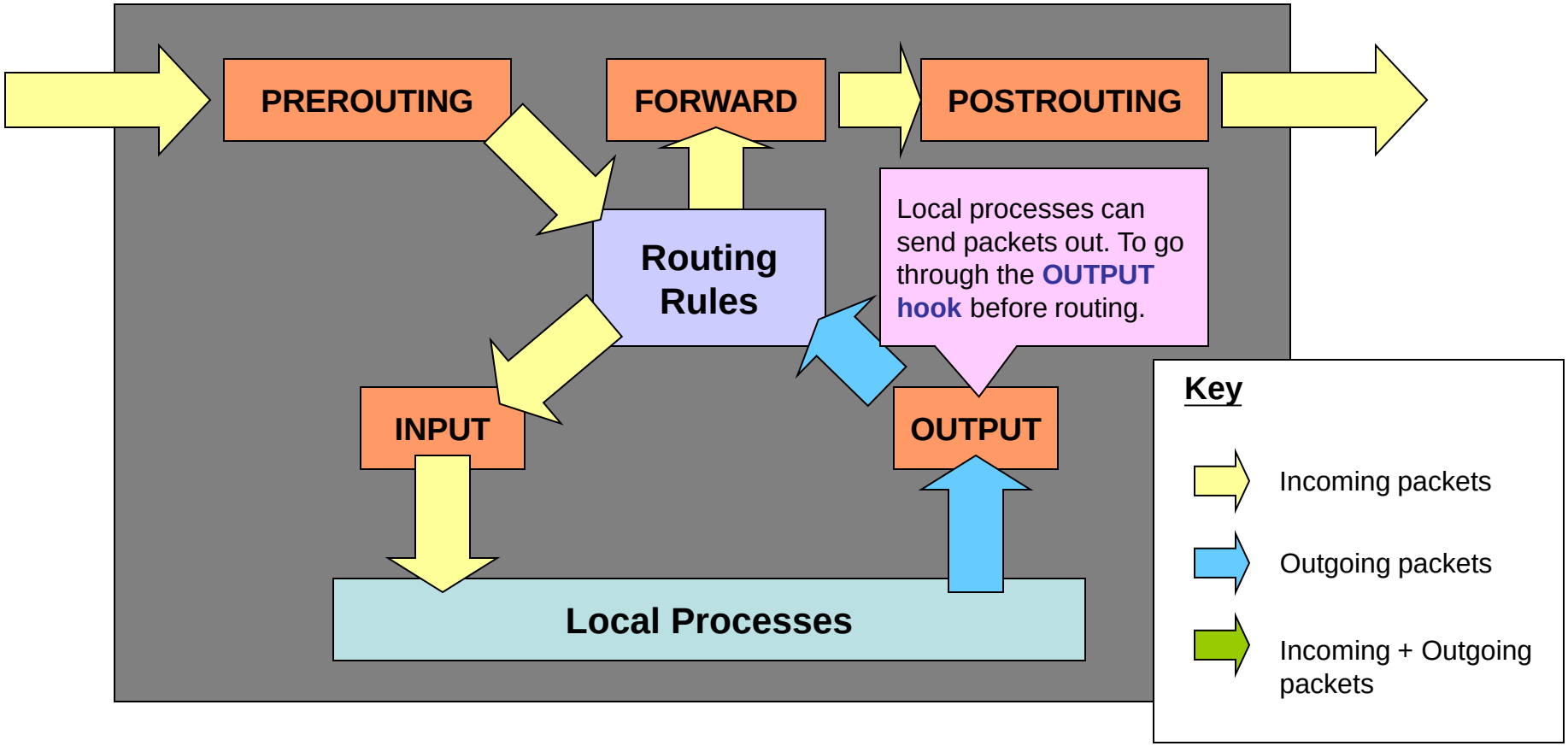
iptables – Packet Flow



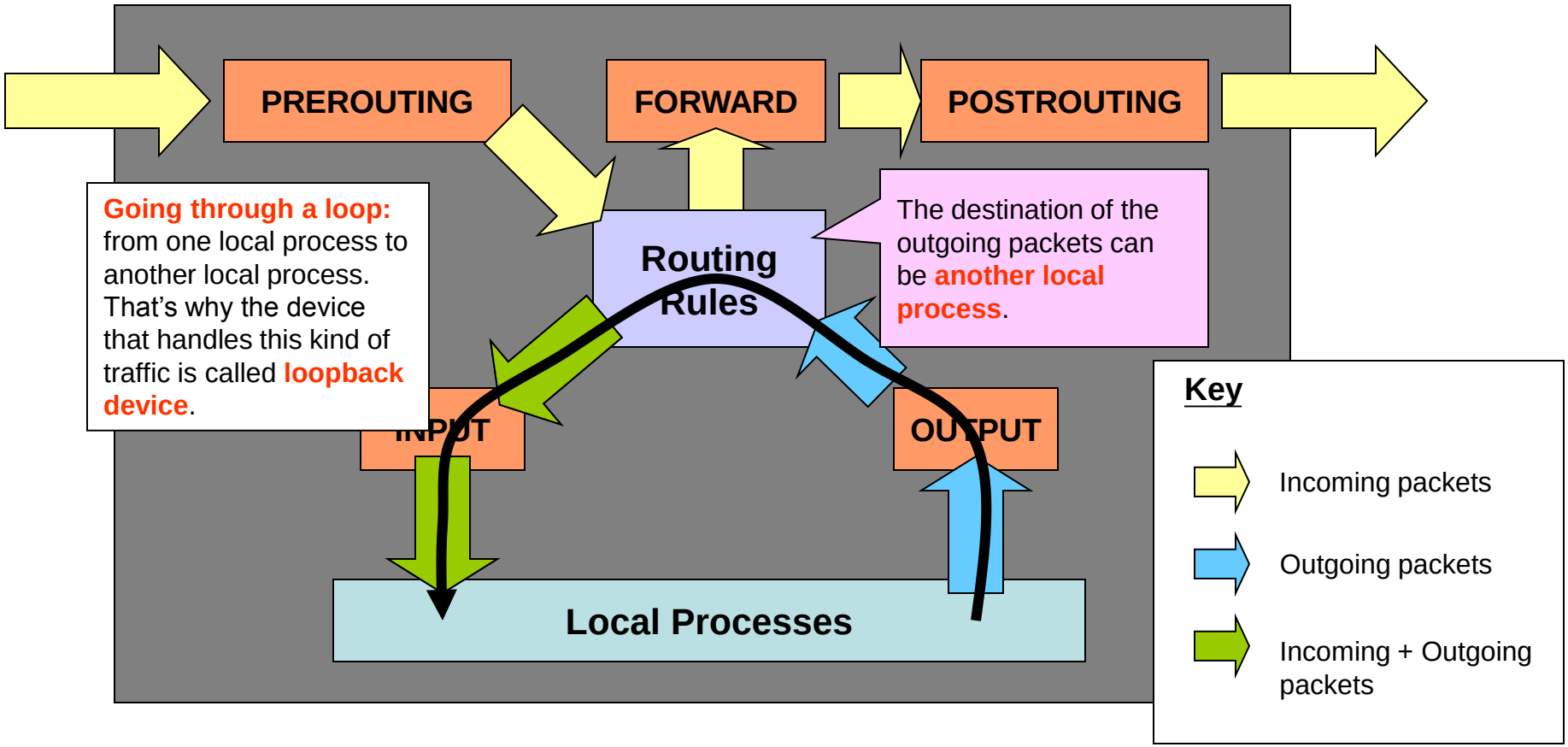
iptables – Packet Flow



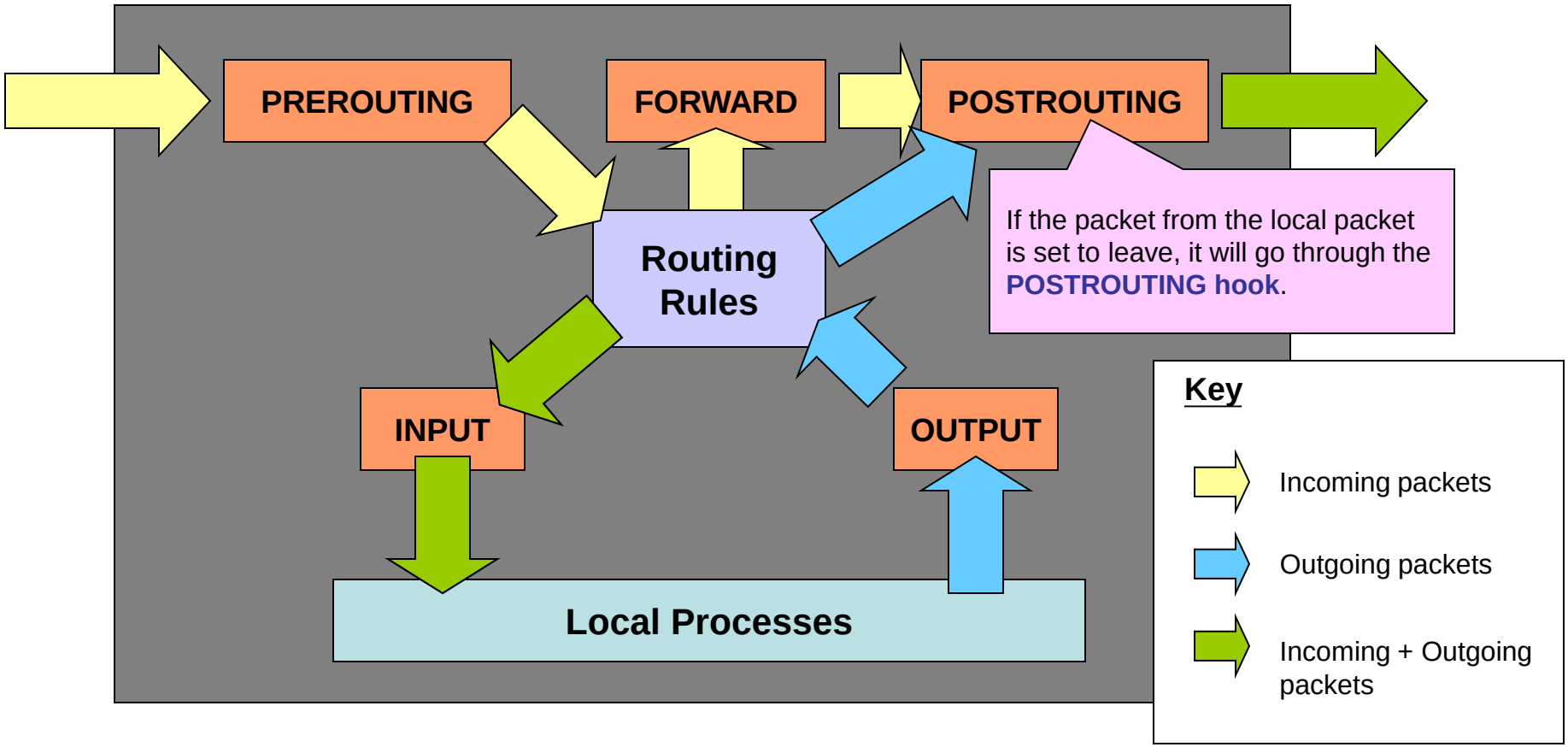
iptables – Packet Flow



iptables – Packet Flow



iptables – Packet Flow



Examples of Using iptables

- List all existing rules (default is filter table):

```
% iptables -L
```

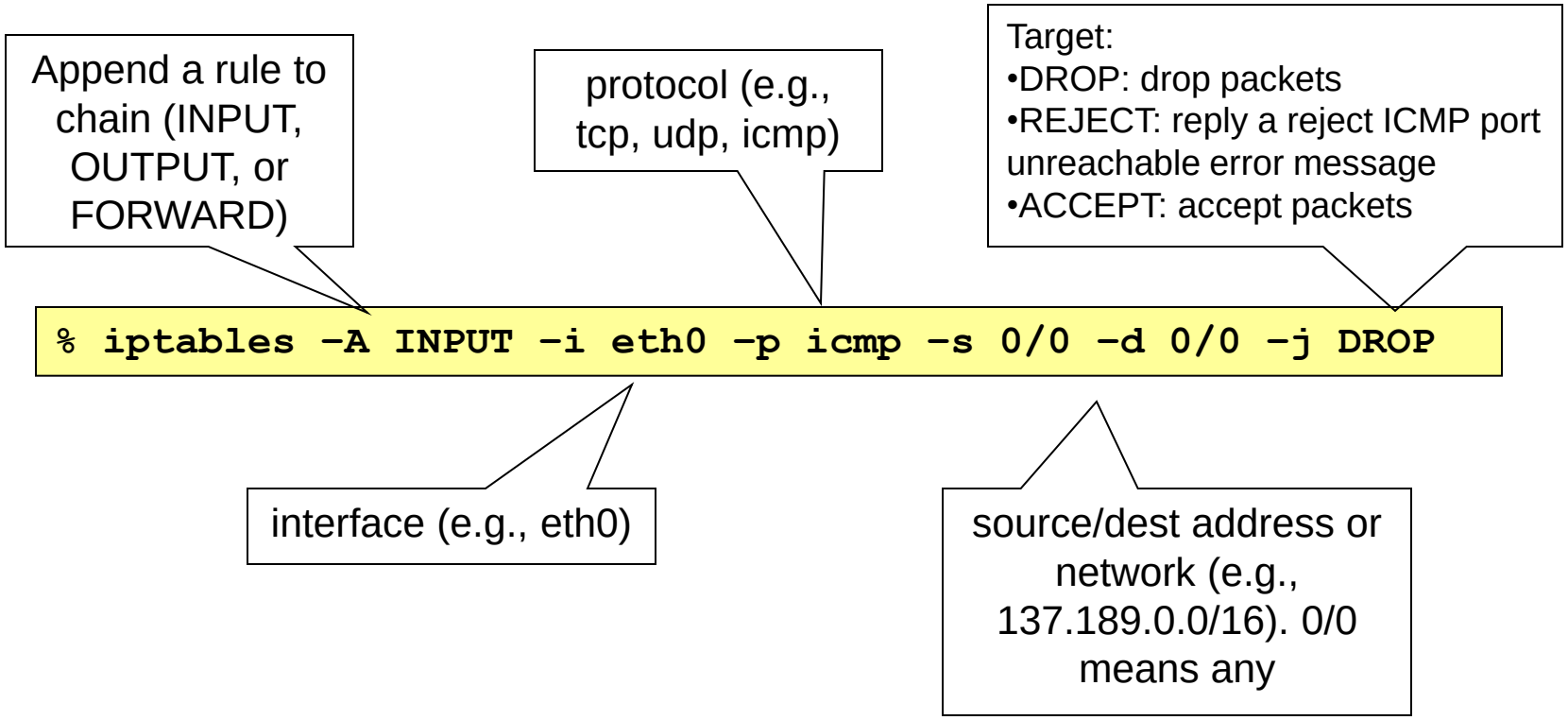
- Flush all existing rules:

```
% iptables -F
```

- Drop all incoming packets to eth0:

```
% iptables –A INPUT –i eth0 –p icmp –s 0/0 –d 0/0 –j DROP
```

Examples of Using iptables



Stateless Packet Filtering

- Accept the web service from CSE machines

```
% iptables –A INPUT –p tcp –s 137.189.88.0/22 -d 0/0 --dport 80 –j ACCEPT
```

- Reject all incoming TCP traffic with flags SYN, SYNACK, or RST that is destined for ports 0 to 1023

```
% iptables –A INPUT –p tcp –s 0/0 –d 0/0 --dport 0:1023 --syn –j REJECT
```

- Reject all outgoing TCP traffic except the one destined for 137.189.5.7

```
% iptables –A OUTPUT –p tcp –s 0/0 –d ! 137.189.5.7 –j REJECT
```

Stateful Packet Filtering

- Stateful inspection with option –m
- Use “-m limit” for SYN flood protection (limit rate to 1/s)

```
% iptables -A FORWARD -p tcp --syn -m limit --limit 1/s -j ACCEPT
```

- Use “–m state” to drop SYN/SYNACK/RST packets that will NOT create new connections

```
% iptables –A INPUT –p tcp --syn –m state ! --state NEW –j DROP
```

Attacks on Firewall

- IP address spoofing
  - fake source address to be trusted
- source routing attacks
  - attacker sets a route other than default to bypass the firewall
- tiny fragment attacks
  - split header info over several tiny packets
  - if the firewall doesn’t do packet reassembly, its rules will become useless

Limitations of Firewall

- Internal attacks cannot be protected
- Single point of failures
- Performance bottleneck

Roadmap

- Firewall
- NIDS (Network Intrusion Detection System)

NIDS

- NIDS is to passive monitor malicious events in network traffic
- If a bad event is identified, alert the operators, or log the event.
- It’s passive, no denying bad requests
  - won’t affect existing traffic

Types of Intrusion Detection

- signature-based
  - Looking for attack patterns
  - e.g., at least K scans in T seconds, or payload has string “0x02030441”
  - need to frequently update attack signatures
- anomaly-based
  - finding abnormal behavior from “normal” behavior from profiling
  - usually based on statistical techniques
  - can have high false alarms

Design Goals of NIDS

See [Paxson, 98]

- High-speed, large volume monitoring
  - packet processing speed should match link capacity
- No packet filter drops
  - should have a large buffer to handle traffic spikes
- Real-time notification
  - An attack should be located ASAP
- Mechanism separate from policy
  - policies can be flexibly changed without re-implementing the system

Design Goals of NIDS

- Extensible
  - Upgrade should be easy
- Avoid simple mistakes
  - policy definitions should be clear
- The monitor will be attacked
  - Our design should prepare that an attacker can know everything about the logic of our system
- Examples of NIDS: Bro, Snort

Snort

- **Snort** is an open-source libpcap-based packet sniffer and logger for lightweight network intrusion detection
  - http://www.snort.org
- Developed by Marty Roesch in 1998
- Design features of Snort
  - lightweight (cross platform, small source)
  - rule-based detection
  - stateful packet analysis
  - extensible

Getting Started with Snort

- Download tarball from http://www.snort.org
  - Need to have PCRE (Perl Compatible Regular Expressions) installed
    - `apt-get install libpcre3 libpcre3-dev`
  - The version that we use is 2.8.6.1.
- Installation is straightforward:
  - Untar the tarball
  - change to the snort source directory
  - ./configure, ./make

What can Snort do?

- Sniffer mode: reads packets off the network and displays them on the console
- Packet logger mode: logs packets to disks
- Network intrusion detection system mode: analyze packets for matches against user-defined rules and perform actions
- Inline mode: obtains packets from iptables and causes iptables to take actions

Snort as a Sniffer

- Making Snort as a sniffer:

```
% snort –v –i eth0
```

- This enables the verbose mode of Snort. Snort will listen to interface eth0. This only prints IP and TCP/UDP/ICMP headers
- You can see the application data with -d

```
% snort –vd –i eth0
```

Snort as a Packet Logger

- You can record packets into disk with the "-l" option (e.g., in directory log/):

```
% snort –v –i eth0 –l ./log
```

- The log file is in pcap format. You can read the log file with "-r" option:

```
% snort –v –r log_file
```

Snort as an NIDS

- You can apply rules to decide what actions to be made to each packet, online or offline.

```
% snort -d -i eth0 -l ./log -c snort.conf     # read from eth0 online
% snort -d -r pkt.pcap -l ./log -c snort.conf # read from file offline
```

- snort.conf is the name of the **rule file**
- In online intrusion detection, the –v switch should be left off for speed.

Snort Alerts

- Alerts are mainly used to log events of interest in NIDS
- When you run snort, include –A switch
- Common options:
  - -A fast: fast alert mode. Write alerts in simple format
  - -A full: full alert mode (default)
  - -A none: turn off alert mode

Writing Snort Rules

- Snort rules are in single line (add \ to the end of line if a rule spans multiple lines).

```
alert tcp $BAD any -> $GOOD any | (flags: SF; msg: "SYN-FIN Scan";)
```

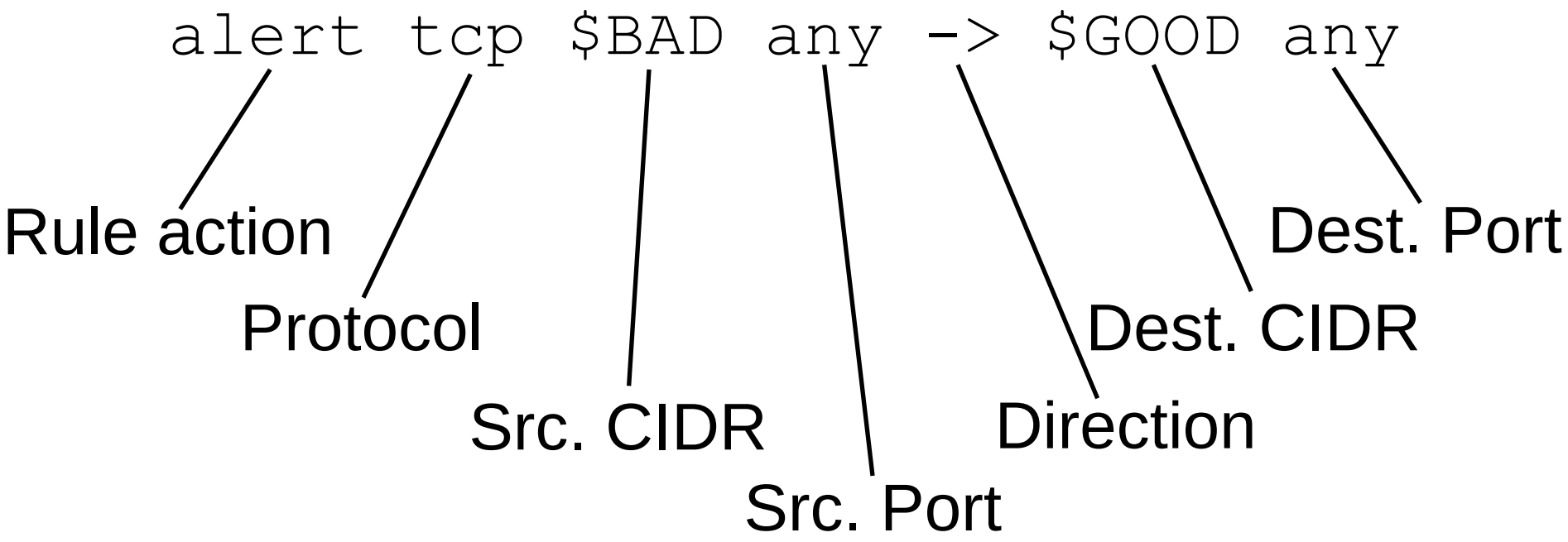
Rule Header

Rule Options

Rule Header
- static definition
- has to be in every rule

Rule Options
- variable definition
- not always necessary

Snort Rule Headers



Snort Rule Options

```
(flags: SF; msg: "SYN-FIN scan";)
```

Option start/finish

Option Detail

```
flags: SF;
```

Keyword

Separator

Argument

Delimiter

Rule Actions

- The rule action tells Snort what to do when it finds a packet that matches the rule criteria
- Example rule actions:
  - alert: generate an alert and logs the packet
  - log: log the packet
  - pass: ignore the packet
  - activate: alert and turn on another rule
  - dynamic: remains idle until activated by an activate rule, then acts as a log rule

Simple Snort Rules

- e.g., record all TCP traffic inbound for port 79 (finger) going to 10.1.1.0/24

```
log tcp any any -> 10.1.1.0/24 79
```

- e.g., port range and bi-directional traffic

```
log tcp 192.168.1.0/24 0:1023 <> !192.168.1.0/24 any
```

- There is no <- operator

Simple Snort Rules

- You can define variables in rules using var
- Example: detect traffic that goes to CUHK

```
var HOME_NET 192.168.1.0/24
var CUHK_NET 137.189.0.0/16
alert tcp $HOME_NET any -> $CUHK_NET any
```

Snort Rules with Options

- Look for HTTP packets that have substring "/cgi-bin/phf" in payload, and output an alert with message "PHF probe!"

```
alert tcp any any <> any 80 \
   (content: "/cgi-bin/phf"; msg: "PHF probe!"; sid: 1000001)
```

  - You can match a byte pattern:
    - content: "E8C0 FFFF FF|/bin/sh";
  - sid specifies the rule ID
    - >1000000 means user-specific rule ID

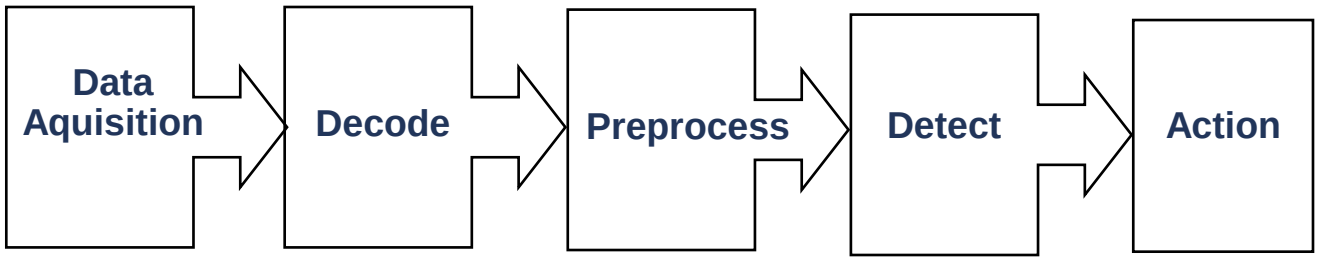
Snort Rules with Options

- pcre enables payload inspection with Perl compatible regular expression

```
alert tcp any any -> any any \
   (pcre: "/^GE*/i"; msg: "HTTP GET"; sid: 1000000)
```

Flow of Snort

- Life of a packet inside Snort
  - **Decode**: parse a packet (in byte stream) from libpcap and specify the packet structure
  - **Preprocess**: advanced decoding, attack detection (will be discussed shortly)
  - **Detect**: checks each packet against the various options listed in the Snort rules files (i.e., for rule matching). Each of the keyword options is a plugin.



Preprocessor

- Rule-matching design aims at matching packet data against signature patterns
  - Do packet by packet independently
  - Stateless approach, not trying to correlate two packets
- **Preprocessors** are modules that enable us to do stateful anomaly detection
  - Non-rule based
  - can correlate behavioral patterns of many packets

Preprocessor

- For example, call an existing preprocessor **`sfportscan`** that detects portscan events
- Create a rule file that has the following:

```
preprocessor sfportscan:
   proto { all }
   scan_type { all }
   sense_level { low }
```

Preprocessor

- Snort provides preprocessors for detecting ARP spoofing

```
preprocessor arpspoof
preprocessor arpspoof_detect_host: 10.0.1.7 00:0C:29:A6:F3:DD
preprocessor arpspoof_detect_host: 10.0.1.10 00:0C:29:AA:BB:CC
```

- Sample output:

```
    10/02-11:07:04.360674  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [*
    10/02-11:07:04.360719  [**] [112:4:1] (spp_arpspoof) Attempted ARP cache overwrite attack [*
…
```

Write your Own Preprocessor

- Snort implements preprocessors as **plug-in modules**, and allows you to add new preprocessors in a systematic way
- Motivations of adding new preprocessors:
  - try your new research ideas
  - work with existing detection modules

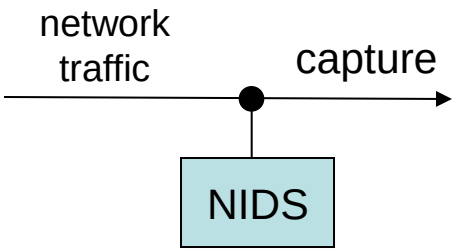
Adding a Preprocessor

- See templates spp_mydetect.*
- Functions a new preprocessor should do:
  - Setup: register the preprocessor keyword and initialization function
  - Init: parse arguments and link other functions
  - Main: process packets and do state tracking
  - Cleanup: cleanup when Snort is exiting

Adding a Preprocessor

- How to include spp* into Snort?
  - Insert an include directive in plugbase.c
  - Insert a call to your plugin Setup() in RegisterPreprocessors()
    - Snort keeps a list of preprocessors and executes them one by one, if specified in the config file.
  - Add your plugin code and header file to Makefile.am and Makefile.in under the preprocessors/ directory
  - Run ./configure and make again
- Demo.

Deploying NIDS



- We need to a packet capture device to collect and duplicate packet copies to the NIDS
- Ideally, we shouldn't drop packets
- Hub:
  - a half-duplex device that repeats a packet on every output interface
  - a device can see all traffic by enabling promiscuous mode
  - slow capture (hence high packet drops)

R. Bejtlich, "Extrusion Detection", Chapter 4.

Deploying NIDS

Cisco Catalyst 2960 series switches

➢ Switch:

- Known as port mirroring
- Has a SPAN (switched port analyzer) port that mirrors traffic received on other ports
- Full duplex mode
- Faster than hubs, but may not fast enough for Gigabit networks
- Product example: Cisco Catalyst Switches, Netgear switches

Deploying NIDS

DAG 9.2X2 10Gb card

- Tap / Capture card:
  - A device specifically for packet capture
  - Has its own packet ring buffer to buffer traffic bursts
  - full duplex mode
  - Either internal (e.g., PCI cards) or external devices
  - Product examples: Endace capture cards, Net Optics taps

References

- Firewall
  - Stallings, Ch. 22
  - iptables tutorial, http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO-7.html
- NIDS
  - V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time”, USENIX 1998.
  - SNORT Users Manual 2.8.6, Ch. 1 and 3.
  - Brian Caswell, “Harnessing the Power of Snort”