

# Unix System Administration

**Chris Schenk**

**Lecture 09 – Thursday Feb 15**

**CSCI 4113, Spring 2007**

# Transport Layer (3)

- Port numbers are 16-bit unsigned int values
  - Used to differentiate applications and services
    - IP address is the **host (device)**, Port is the **process**
  - Transport layer protocols must use both source and dest ports
  - Important services use well-known port numbers
    - Usually found within /etc/services
    - Very long list full of ports you've never heard of
  - Restricted ports (ports < 1024) require root
    - *Very mild security*

# Transport Layer - UDP

- User Datagram Protocol
  - **Unreliable** and **stateless** like IP
    - Applications must write-in reliability if needed
  - Good for short, one-time things like
    - NTP
    - DNS queries
    - Streaming video (lots of packets, but each small)

<b>16 bit SOURCE port number</b>	<b>16 bit DESTINATION port number</b>
<b>16 bit length (incl hdr) in bytes</b>	<b>16 bit hdr chksm – unused</b>

# Traceroute

- Used to 'map' routes between you and the destination
- Traceroute sends arbitrary UDP packets to dest
  - Dest port is set to an unlikely value
  - IP time-to-live (TTL) field is incremented at each hop
  - ICMP time exceeded (11) error returned from routers
  - ICMP port unreachable (code 3) returned from dest

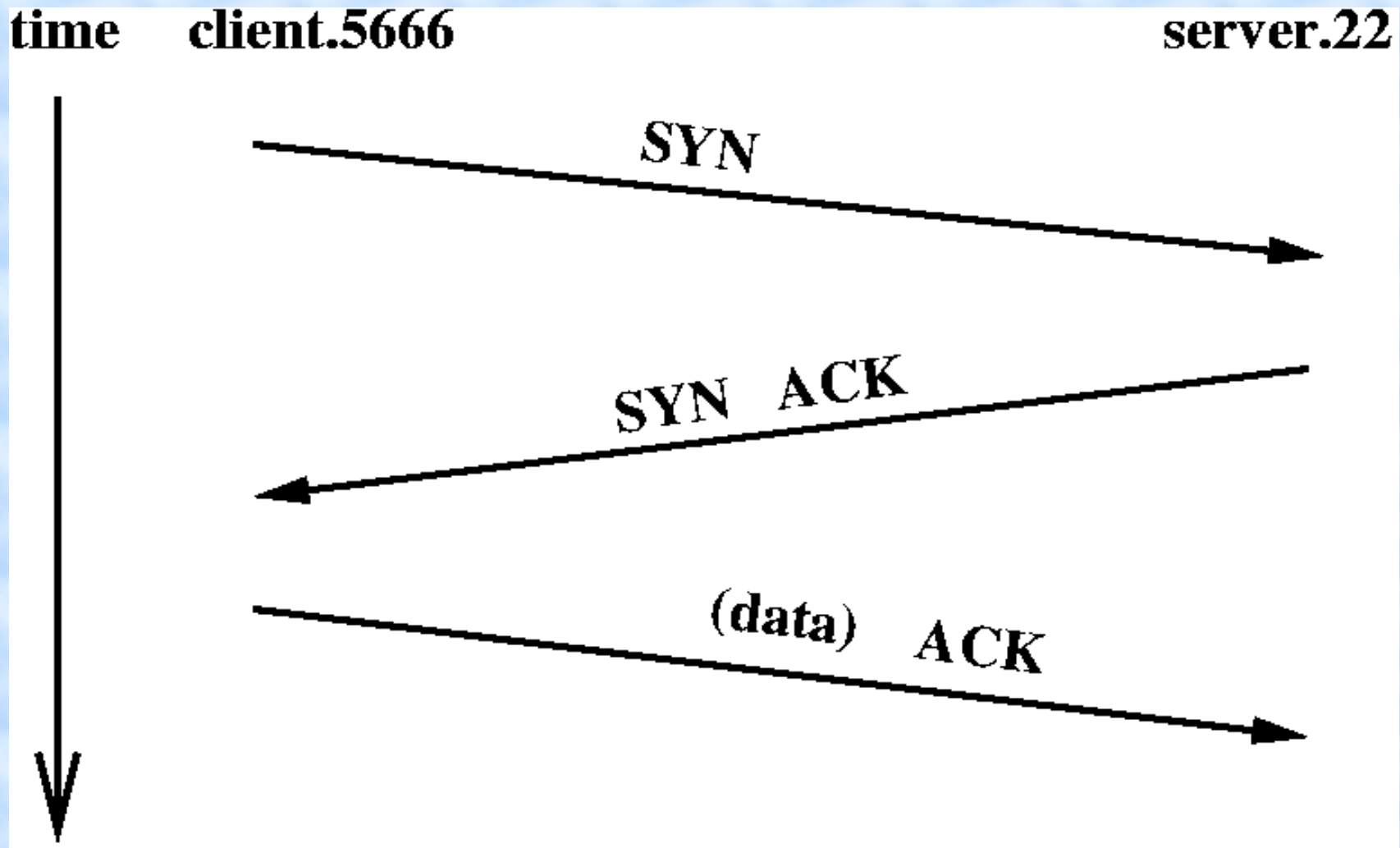
# Transport Layer (3) – TCP

- Transmission Control Protocol
  - Both reliable AND stateful
  - Reliability is done with timeout and re-transmission
  - A connection is defined by a source IP address, port-number pair and a destination IP address, port-number pair
  - Full Duplex means data flows in both directions (accomplished using piggyback ACKs)

# Transport Layer – TCP Header

<b>16 bit Source Port</b>		<b>16 bit Destination Port</b>	
<b>32 bit Sequence Number</b>			
<b>32 bit Acknowledgement Number</b>			
<b>4 bit header length</b>	<b>6 bit reserved</b>	<b>6 bit flags</b>	<b>16 bit Window</b>
<b>16 bit Checksum</b>		<b>16 bit Urgent Pointer</b>	
<b>Options (if any)</b>			

# Transport Layer – TCP Handshake



# Transport Layer (3) – TCP (cont)

- Other features of TCP
  - MSS – Maximum Segment Size (65507 bytes)
    - similar to MTU
  - Delayed ACKs (piggybacking)
    - Sending acks with data
  - Sliding window
    - More efficiency in sending data
  - Network congestion avoidance
  - and others

# Ephemeral Ports

- A connection requires the endpoint IP-Port pair
  - Connect to google.com at port 80
- Client browser doesn't have to explicitly bind to a local port
  - This is where ephemeral ports come in
- Ephemeral ports range varies between systems
  - Linux: `/proc/sys/net/ipv4/ip_local_port_range`
    - 32768-61000
  - FreeBSD: `sysctl -a | grep net.inet.ip.portrange`
    - 1024-5000

# TCPDump

- Used to display TCP/IP packet headers
  - usually as directly received from an active network via an interface in PROMISCUOUS mode
  - Packets are specified by specifying a boolean expression
- Useful options:
  - `-i <interface>` listen on a specific interface
  - `-e` give link-layer header also
  - `-n` give IP addresses (no DNS)
  - `-t` don't give timestamps
  - `-v` give more header fields (verbose)

# TCPDump (cont)

- TCPDump boolean expressions
  - If none given, all packets are displayed
  - Expressions consist of primitives combined with **and, or, not**
    - large variety of primitives (see the manpage)
    - parentheses may be used (but must be escaped)
  - Example primitives:
    - `src host csel`
    - `port ftp and not port ftp-data`
    - `dst port http and host csel`

# Ethereal (Wireshark)

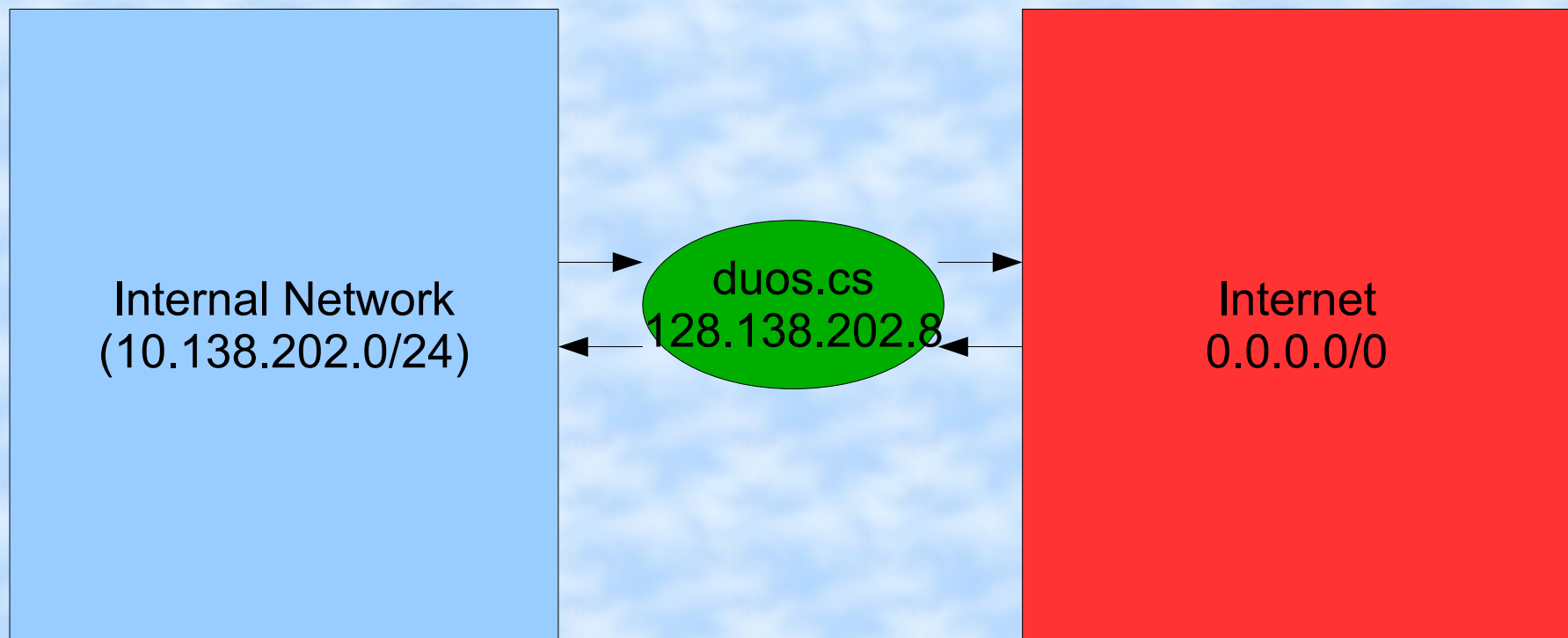
- Graphical packet sniffing tool
  - same behavior as tcpdump, but somewhat easier to see
  - decodes and splits packets into readable formats
    - Link, Network, Transport, and Application layers
- Reads capture files from tcpdump
  - Also uses the same boolean expressions

# Netstat

- Has many uses
  - View the system's routing table
  - View active or listening TCP and UDP services
  - Other network stats
  - Implementation varies by OS
- Some options
  - `-i` Show stats for all interfaces
  - `-a` Show server socket state too
  - `-n` Don't do IP->name (DNS) lookups
  - `-r` Show system's routing table
  - `-vaten` I find this one most useful

# Network Address Translation (NAT)

- NAT provides a way to mask machines on a network
  - Single IP address exposed for multiple hosts



# Network Address Translation (cont)

- The SOURCE IP address and port is rewritten to the gateway's IP address and port
  - Kyle is 10.138.202.72 and ephemeral port 37019
  - Rewritten to 128.138.202.8 and ephemeral port 41000
- The gateway machine waits for a response from server
  - Looks at the rewritten ephemeral response to know how to rewrite to the original IP and port of Kyle
  - So any incoming packets for ephemeral port 41000 are rewritten to IP 10.138.202.72 and port 37019 for Kyle

# Layer 2/3 Firewalls

- NOT to be confused with Proxies
  - Provide a service, content filtering, caching
    - http proxies (web, aim), ftp proxies
- Stateful Packet Inspection
  - Smart firewalls know about all protocols and expected packets
  - CPU and RAM intensive to maintain information
- Rule sets, ordering, and policies
  - Differ for many types of firewalls

# Rule Order

- IP firewalls consist of ORDERED lists of rules
- Some firewalls operate as “first match wins”
  - The first rule to match a specific packet immediately determines its fate
  - These firewalls must be ordered from *MOST* to *LEAST* specific in terms of matching rules
  - For example, a BAD ordering would be:
    - block from any to any tcp | allow from any to any tcp port 22
  - While a GOOD ordering is:
    - allow from any to any tcp port 22 | block from any to any tcp

# Rule Order (cont)

- Other firewalls run on a “last match wins” basis
    - The last rule in a set to match a given IP packet determines its fate
    - These must be ordered from *LEAST to MOST* specific, the exact opposite of first-match firewalls
    - Uses more system resources
      - Runs through more rules!
    - Often can be short-circuited to behave like first-match firewalls
    - A GOOD ordering would be
      - block from any to any tcp | allow from any to any tcp port
- 22

# Default Policy

- Determines what to do with packets that do not match any given rules
- Safest policy is to “block all incoming traffic”
- Often sites don't filter outgoing traffic
  - Some people consider this to be a security risk
- Depending on rule order processing, the default policy is expressed either at the beginning or end of the ruleset
  - last-match-wins express policy FIRST
  - first-match-wins express policy LAST

# Rule Groups

- Some firewalls offer rule grouping
- One rule is used to trigger a group
  - If packet matches the trigger rule, then all rules in the group are run against the packet
  - Otherwise the group of rules is ignored
  - User-defined chains in IPTables are these groups
- Useful for optimizing firewalls on busy gateways
  - Create the shortest rulesets possible
  - More rules == longer processing time

# Traffic Direction

- Firewall rules are usually expressed in terms of SOURCE and DEST values found in packet headers
  - e.g. source IP address, dest TCP port number, etc
- SOURCE and DEST are **relative** to traffic direction!
  - TCP 'syn' packet sent to server
  - TCP 'synack' sent back to client (reverse dir.)
  - TCP 'ack' sent back to server (original direction)

# Firewalls on Different Systems

- Solaris 10, FreeBSD, NetBSD
  - IPFilter by Darren Reed, last-match-wins
- OpenBSD
  - PF – Packet Filter, last-match-wins
  - My favorite firewall!
  - Very simple and straightforward rule creation
- Linux
  - IPTables aka NetFilter, first-match-wins

# IPTables – Tables and Chains

- What are these 'tables'?
  - Raw, mangle, nat, filter, (and others)?
- Every table has built-in chains
  - Not all tables have the same chains!
- Raw handles all incoming traffic
  - PREROUTING and OUTPUT chains for 'raw'
- Mangle next, potentially modifying packets
  - TOS (type of service), TTL (time to live)
- Filter for doing just that, filtering.

# Filter Table – INPUT Chain

- For all incoming traffic
- When a packet is determined to be destined for this machine, the packet is sent to the INPUT chain
  - Packet determined to be for this machine by:
    - Dest MAC address is broadcast (FF:FF:FF:FF:FF:FF)
    - Dest IP address is this machine
    - Dest IP address is broadcast (host bits all ones - 1)
- If the packet is **ACCEPT**ed, the packet is sent to the proper process
  - Otherwise the packet is dropped

# OUTPUT and FORWARD Chains

- FORWARD chain also for incoming traffic
- If forwarding is enabled, and the packet is destined for another network interface, the packet is sent to the FORWARD chain
  - If ACCEPTed, the packet is sent to the other network interface
- When a process wants to send a packet out, the packet is sent immediately to the OUTPUT chain
  - If ACCEPTed, then packet is sent to the network interface

# Using IPTables – Chains

- There are operations for manipulating chains and also rules within chains (manpage is super-detailed)
- Some chain manipulation options
  - Create new chain (-N)
  - Delete empty chain (-X)
  - Change policy for chain (-P)
  - List rules for a chain (-L)
  - Flush rules out of a chain (-F)
  - Zero packet and byte counters for all rules in a chain (-Z)

# Using IPTables – Rules

- Several options to manipulate rules within chains
  - Append new rule to chain (-A)
  - Insert new rule at a position within a chain (-I)
  - Replace a rule at a position within a chain (-R)
  - Delete a rule at a position within a chain (-D)
  - Delete the first rule that matches within a chain (-D)
- Most common used are -A and -D
  - -I and -R are merely extensions of append and delete

# Using IPtables – Rule Parameters

- Parameters help clarify certain actions on rules
  - Source (-s)
  - Destination (-d)
  - Protocol (-p)
  - Jump (-j)
  - Input-interface (-i)
  - Output-interface (-o)

# Using IPtables – Rules (cont)

- Rules are the bread and butter of any firewall
- Each rule specifies conditions for packet
- Rules also specify actions after condition is met
  - Called a “target” in iptables
    - Target types – ACCEPT, DROP, QUEUE, RETURN, or other chains
- Usage is fairly simple
  - iptables -A <chain> -s <source> -p <protocol> -j <target>
  - iptables -D <chain> <rule number | full rule>

# Using IPtables – Simple Example

- We want to drop all incoming icmp packets to the loopback interface (lo – 127.0.0.1)
- Append DROP target to rule
  - iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
- Delete rule by stating entire rule
  - iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
- Delete rule by stating index of rule within chain
  - iptables -D INPUT 1

# Using IPtables – Filtering Specifications

- Option '-s' used for source IP address
- Option '-p' used for protocol
- Also have '-d' for destination IP
- Four ways to specify IP addresses
  - Two for individual hosts, two for groups of hosts
  - By name – 'localhost' or 'google.com' (BAD!)
  - By IP address – 127.0.0.1 or 128.138.202.19
  - By CIDR mask – 128.138.202.0/24
  - By Network/Mask – 128.138.202.0/255.255.255.0

# Using IPtables – Subnet Example

- Let's drop all outgoing ICMP packets to the 202 subnet
  - iptables -A OUTPUT -d 128.138.202.0/24 -p icmp -j DROP
- Let's replace the rule to include all of CU
  - iptables -R OUTPUT 1 -d 128.138.0.0/16 -p icmp -j DROP
- Delete the rule
  - iptables -D OUTPUT 1

# IPTables - Saving/Restoring

- Speedy saving/loading of rules
  - **iptables-save, iptables-restore**
  - `iptables-save [-c] [-t table] > /tmp/myfirewallrules.txt`
  - `iptables-restore [-c] < /tmp/myfirewallrules.txt`
- Scripts can invoke iptables individually for rules
  - Very slow compared to a complete save/load
  - Each add/remove loads the entire ruleset into the kernel
- No scripting available within the iptables files

# IPTables – Implicit Matches

- Matches know how to track specific protocol types
- TCP, UDP, ICMP
  - These three have implicit match types
    - A `-m tcp` is not required
  - `% iptables -A INPUT -p TCP`
  - Parameters for each type
    - TCP: `--sport`, `--dport`, `--tcp-flags`, `--tcp-option`
    - UDP: `--sport`, `--dport`
    - ICMP: `--icmp-types`

# IPTables – Port Filtering

- Using the Layer 3 parameters
  - iptables -A INPUT -p tcp --dport 22 -j DROP
  - iptables -A INPUT -p tcp --sport ! 30:1023 -j DROP
    - **Note:** the ! sign
  - iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
  - iptables -A INPUT -p icmp --icmp-type 8 -j ACCEPT
  - iptables -A OUTPUT -p icmp --icmp-type 11/1 -j DROP
- Note the **11/1** for specifying ICMP codes

# IPTables – Explicit Matches

- Requires a **-m** or **--match** parameter
- Some matches protocol specific, some not
- Each match type provides more parameters
- **state** matches require `-m state` before matching a connection state, followed by a `--state` parameter
  - `iptables -A INPUT -m state --state RELATED,ESTABLISHED`
- **mac** states allow for `--mac-source`
  - `iptables -A INPUT -m mac --mac-source 01:23:45:67:89:AB`

# IPTables – Limit Module

- Completely non-intuitive
  - Intuitive would be: Drop  $X$  packets that match rule  $R$  per  $Y$  time unit
  - IPTables is not this.
- `-m limit --limit 5/s --limit-burst 10`
  - Limit-burst token bucket set to 10
    - Each packet matched depletes one token from bucket
  - The `5/s` means restore 1 token to the bucket every  $1/5$  second
  - Once token bucket is empty, rules no longer match!