Goals of Today’s Lecture

- Network neutrality
- Elements of control: **Middleboxes**
  - Network address translators (NATs)
  - Firewalls
  - Tunneling
  - Application gateways
- Middleboxes introduce new problems while solving existing ones
  - Erosion of end-to-end semantics
  - Connections become more brittle
  - New apps harder to deploy (impairs innovation)
Network Neutrality

- *Network neutrality* = notion that ISPs supply non-discriminatory IP connectivity
- Opposite counterpoint: a **Walled Garden**
  - Provider only allows you to access their (often value-added) services
  - E.g., AOL's captive Web portal/email/IM
- **Highly contentious; potential legal fray**
  - E.g., ISPs blocking Voice-over-IP (VOIP)
    - As does China, Panama, Costa Rica
  - E.g., ISPs blocking 25/tcp (SMTP) to curb spammers
  - E.g., Verisign's attempted *SiteFinder service*
    - For DNS typos, returned page "Would you like to buy this domain?"
  - E.g., ComCast's (imperfect) blocking of BitTorrent
Network Neutrality, con’t

- Is Internet access something that ISPs provide as “common carriers”?
  - Transporting goods as service to common public
- Or: will free market forces serve to shape ISP favoritism efficiently?
  - Is there a danger of monopolies emerging?

Network-Layer Principles

- Globally unique identifiers
  - Each node has a unique, fixed IP address
  - … reachable from everyone and everywhere
- Simple packet forwarding
  - Network nodes simply forward packets
  - … rather than modifying or filtering them
Internet Reality

- Host mobility
  - Changes in IP addresses as hosts move
- IP address depletion
  - Dynamic assignment of IP addresses
  - Use of private addresses
- Security/policy concerns
  - Discarding suspicious or unwanted packets
  - Monitoring activity
- Performance concerns
  - Storing popular content near the clients
  - Network neutrality, or lack thereof

Middleboxes

- **Middleboxes** are intermediaries
  - Interposed in-between the communicating hosts
  - Often without knowledge of one or both parties
    - “hidden” / “transparent”

- Examples
  - Network address translators (**NATs**)
  - Firewalls
  - Performance boosters
  - Intrusion detection/prevention systems
  - Transparent Web proxy caches
  - Sign-on “capture” pages
Two Views of Middleboxes

- An abomination
  - Violation of layering
  - Cause confusion in reasoning about the network
  - Responsible for many subtle bugs

- A necessity
  - Solving real and pressing problems
  - Needs that are not likely to go away

Network Address Translation (NAT)

Review (see lecture 7 on IP Addressing)
Active Component in the Data Path

IP Header Translators

- Local network addresses not globally unique
  - E.g., private IP addresses (in 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16; see RFC 1918)
- NAT box rewrites IP addresses
  - Make the “inside” look like a single IP address
  - … and change header checksums accordingly
- Outbound traffic: from inside to outside
  - Rewrite the source IP address
- Inbound traffic: from outside to inside
  - Rewrite the destination IP address
Using a Single Source Address

What if Both Hosts Contact Same Site?

- Suppose hosts contact the same destination
  - E.g., both hosts open a socket with local port 3345 to destination 128.119.40.186 on port 80
- NAT gives packets same source address
  - All packets have source address 138.76.29.7
- Problems
  - Can destination differentiate between senders?
  - Can return traffic get back to the correct hosts?
- How can we fix this?
Port-Translating NAT

- Map outgoing packets
  - Replace source address with NAT address
  - Replace source port number with a new port number
  - Remote hosts respond using (NAT address, new port #)
- Maintain a translation table
  - Store map of (source address, port #) ⇔ (NAT address, new port #)
- Map incoming packets
  - Consult the translation table
  - Map the destination address and port number
  - Local host receives the incoming packet

Network Address Translation Example

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80
2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table
3: Reply arrives dest. address: 138.76.29.7, 5001
4: NAT router changes datagram dest. addr from 138.76.29.7, 5001 to 10.0.0.1, 3345
Maintaining the Mapping Table

- Create entry upon seeing outbound packet
  - Packet with new (source addr, source port) pair
- Eventually, need to delete the map entry
  - But when to remove the binding?
- If no packets arrive within a time window
  - … then delete the mapping to free up the port #s
- Yet another example of “soft state”
  - I.e., removing state if not refreshed for a while
- Makes Internet connectivity more brittle

The Problem is Broader

- Do IP addresses only appear in IP headers?
- Also appear in application payloads to facilitate rendezvous (subsequent conn’s)
  - E.g., http://141.16.9.1/good_stuff.html
  - E.g., PORT 141,16,9,1,4,21 (FTP)

- NAT needs to fix these up too
  - Otherwise the application breaks
- How hard is that?
Modifying Addresses Gets Messy

- Problem #1: what if replacement has different number of bytes than original?
  - Okay, we must adjust TCP sequence numbers
  - And: rewrite ACKs
- Problem #2: what if revised packet > MTU?
  - Um, send multiple pkts (or allow fragmentation)
- Problem #3: what if we don’t know where in the payload the app embeds addresses?
  - Oops: the app won’t work through the NAT
- NATs make it harder to deploy new apps

Servers Behind a NAT

NATs undermine using port #s to address processes
NAT needs binding in advance for incoming SYNs
Objections Against NAT

- Difficult to support peer-to-peer applications
  - P2P needs a host to act as a server
- Layering violation (hence messiness)
- NAT violates the end-to-end principle
  - Network nodes should not modify the packets
- Connections become brittle
- Barrier to deployment of new apps
- IPv6 is a cleaner solution
  - Better to migrate than to limp along with a hack

5 Minute Break

Questions Before We Proceed?
Firewalls

- Isolates organization’s internal net from Internet
- Allows some packets to pass, blocks others
  - (Refinement: shape some traffic, allow other unimpeded)
- Twin goals: security and policy enforcement
Packet Filtering

- Firewall filters packet-by-packet, based on:
  - Source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits
- Simpler versions are stateless
  - But increasingly they need to be stateful

Packet Filtering Examples

- Block all packets with IP protocol field = 17 or with either source or dest port = 22.
  - All incoming and outgoing UDP flows blocked
  - All SSH connections blocked
- Block inbound TCP with SYN but no ACK
  - Prevents external clients from initiating TCP connections to internal hosts
  - But allows internal clients to connect to outside
- Block all packets with TCP port of Halo 3
  - (Oops, let’s hope no other app uses that port)
- Block all packets with “reserved” bits set
Firewall Configuration

- Firewall applies a set of rules to each packet
  - To decide whether to permit or deny the packet
- Each rule is a test on a packet’s header fields
  - Test yields permit or deny
- Order matters: first matched rule wins
- E.g.: Alice runs a network in 222.33.0.0/16
  - Wants to let Bob’s site access certain hosts
    - Bob is on 111.55.0.0/16
    - Alice’s special hosts on 222.33.44.0/24
  - Alice doesn’t trust Trudy, inside Bob’s network
    - Trudy is on 111.55.66.0/24
  - Alice doesn’t want any other traffic from the Internet

Firewall Configuration, con’t

- Alice’s firewall rules
  - #1: Don’t let Trudy machines in
    - Deny <src = 111.55.66.0/24, dst = 222.33.0.0/16>
  - #2: Let rest of Bob’s network in to special dsts
    - Permit <src=111.55.0.0/16, dst = 222.33.44.0/24>
  - #3: Block the rest of the world
    - Deny <src = 0.0.0.0/0, dst = 0.0.0.0/0>
- It’s easy to introduce subtle errors …
  - And production firewalls can have 1000s of rules
Subverting Firewalls

- How can we fool a firewall?
- Method #1: abuse its statelessness
  - Consider the rule of “no SYNs w/o ACKs” as a way to prevent connections initiated inbound
  - How can we mislead a firewall about setting of TCP flag bits?

Checking TCP Header for Initial SYN

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen 0</td>
<td>Advertised window</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

Firewall will check here, i.e., 14 bytes into transport header just after IP header.
### Misleading Stateless Inspection

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Split into two **fragments**. First is just 8 bytes of IP payload, i.e., here.

### Misleading Stateless Inspection, con’t

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Second fragment starts 8 bytes later covering all of this.

Firewall looks 14 bytes into payload, i.e., here, which is **under the control of the attacker**.
Subverting Firewalls, con’t

- How might a firewall defend against this?
  - Defense #1: reassemble fragments
    - But this costs state
  - Defense #2: deny small initial fragments
    - But: legit traffic has these, hence collateral damage

- Subversion Method #2: abuse ports
  - Who says that e.g. port 22/tcp = SSH?
    - Why couldn’t it be say Skype or Bit Torrent?
    - Just requires that client & server agree on app proto

Hiding on Other Ports

- Method #1: use port allocated to another service (how can this be detected?)
- Method #2: tunneling
  - Encapsulate one protocol inside another
  - Receiver of “outer” protocol decapsulates interior tunneled protocol to recover it
  - Pretty much any protocol can be tunneled over another (with enough effort)
- E.g., tunneling IP over SMTP
  - Just need a way to code an IP datagram as an email message (either mail body or just headers)
Example: Tunneling IP over Email

From: doesn't-matter@bogus.com
To: my-buddy@tunnel-decapsulators.R.us
Subject: Here's my IP datagram

IP-header-version: 4
IP-header-len: 5
IP-ID: 11234
IP-src: 1.2.3.4
IP-dst: 5.6.7.8
IP-payload: 0xa144bf2c0102…

Program receives this legal email and builds an IP packet corresponding to description in email body …
… injects it into the network
How can a firewall detect this??

Tunneling, con’t

- E.g., IP-over-ICMP:
  - Encode an IP datagram as the payload of a “ping” packet
- E.g., Skype-over-HTTP:
  - Encode Skype message in URL of requests or header fields (or cookies) of replies
- Note #1: to tunnel, the sender and receiver must both cooperate
- Note #2: tunneling has many legitimate uses too
  - E.g., overlay networks that forward packets along paths different from what direct routing would pick
  - E.g., Virtual Private Networks (VPNs)
    - Make a remote machine look like it's local to its home network
    - Tunnel encrypts traffic too for privacy
Application Gateways

- Middlebox can insert itself between client and server
  - Client deals with middlebox (application gateway), not server
  - Server deals with middlebox, not client
  - Can be done explicitly or transparently
- Example: Web proxy
- Example: SSH gateway
  - Require all SSH in/out of site to go through gateway
  - Gateway logs authentication, inspects decrypted text
  - Site’s firewall configured to prohibit any other SSH access

SSH Gateway Example

Firewall

permit <port=22, host=1.3.5.7>
deny <port=22>

1.3.5.7
Conclusions

- Middleboxes address important problems
  - Using fewer IP addresses
  - Blocking unwanted traffic
  - Monitoring activity
  - Shaping use of network resources
  - Improving/controlling performance (vs. network neutrality)

- Middleboxes cause problems of their own
  - Connectivity erodes
    - Notion of addresses, ports weakened
    - Middlebox state management can lead to connection termination
  - Harder to deploy new apps