CIS 433/533 - Computer and Network Security

Firewalls

Professor Kevin Butler
Winter 2011
Firewalls

- A firewall ... is a physical barrier inside a building or vehicle, designed to limit the spread of fire, heat and structural collapse.
Filtering: Firewalls

- Filtering traffic based on *policy*
  - Policy determines what is acceptable traffic
  - Access control over traffic
  - Accept or deny

- May perform other duties
  - Logging (forensics, SLA)
  - Flagging (intrusion detection)
  - QOS (differentiated services)
IP Firewall Policy

- Specifies what traffic is (not) allowed
  - Maps attributes to address and ports
  - Example: HTTP should be allowed to any external host, but inbound only to web-server

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Flags</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Port</td>
<td>Address</td>
<td>Port</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>1.1.1.1</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>1.1.1.*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>TCP</td>
</tr>
</tbody>
</table>
X-Listing

- **Blacklisting** - specifying specific connectivity that is explicitly disallowed
  - E.g., prevent connections from badguys.com

- **Whitelisting** - specifying specific connectivity that explicitly allowed
  - E.g., allow connections from goodguys.com

- Useful for IP filtering, spam mitigation, …
Stateful, Proxy, and Transparent

- Single packet contains insufficient data to make access control decision
  - **Stateful**: allows historical context consideration
  - Firewall collects data over time
    - e.g., TCP packet is part of established session
- Firewalls can affect network traffic
  - **Transparent**: appear as a single router (network)
  - **Proxy**: receives, interprets, and reinitiates communication (application)
  - Transparent good for speed (routers), proxies good for complex state (applications)
DMZ (De-militarized Zone)

- Zone between LAN and Internet (public facing)
Practical Issues and Limitations

- Network layer firewalls are dominant
  - DMZs allow multi-tiered fire-walling
  - Tools are widely available and mature
  - Personal firewalls gaining popularity

- Issues
  - Network perimeters not quite as clear as before
    - E.g., telecommuters, VPNs, wireless, …
  - Every access point must be protected
    - E.g., this is why war-dialing is effective
  - Hard to debug, maintain consistency and correctness
  - Often seen by non-security personnel as impediment
    - E.g., Just open port $X$ so I can use my wonder widget …
The Wool firewall study..

- 12 error classes
  - No default policy, automatic broad tools
  - NetBIOS (the very use of the Win protocol deemed error)
  - Portmapper protocols
  - Use of “any wildcards”
  - Lack of egress rules

- Interesting questions:
  - Is the violation of Wool’s errors really a problem?
  - “DNS attack” comment?
  - Why do you think more expensive firewalls had a higher occurrence of errors?

- **Take away**: configurations are bad
Practical Firewall Implementations

- Primary task is to filter packets
  - But systems and requirements are complex
- Consider
  - All the protocols and services
  - Stateless vs. stateful firewalls
  - Network function: NAT, forwarding, etc.

- Practical implementation: Linux iptables
Netfilter hook

- Series of hooks in Linux network protocol stack
- An `iptables` rule set is evaluated at each

Hook placements:
iptables Concepts

- **Table**
  - All the firewall rules

- **Chain**
  - List of rules associated with the chain identifier
  - E.g., hook name

- **Match**
  - When all a rule’s field match the packet (protocol-specific)

- **Target**
  - Operation to execute on a packet given a match
iptables Commands

iptables [-t <table_name>] <cmd> <chain> <plist>

- **Commands**
  - *Append* rule to end or specific location in chain
  - *Delete* a specific rule in a chain
  - *Flush* a chain
  - *List* a chain
  - *Create* a new user-specified chain
  - *Replace* a rule
Test it out

- PING on localhost
  - `ping -c 1 127.0.0.1`

- Add iptables rule to block
  - `iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP`

- Try ping

- Delete the rule
  - `iptables -D INPUT 1`
  - `iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP`
  - `iptables -F INPUT`
Testing

- Use loopback to test the rules locally on your machine
  - IP address 127.0.0.1
- ICMP
  - submit ping requests to 127.0.0.1 as above
- TCP
  - submit requests to 127.0.0.1 at specific port
    - server
      - `nc -l -p 3750`
      - listen at port 3750
    - client
      - `nc -p 3000 localhost 3750`
      - send from port 3000 to localhost at port 3750
iptables Rule Parameters

- Destination/Source
  - IP address range and netmask
- Protocol of packet
  - ICMP, TCP, etc
- Fragmented only
- Incoming/outgoing interface
- Target on rule match
Per Protocol Options

- Specialized matching options for rules
  - Specific to protocol
- TCP
  - Source/destination ports
  - SYN
  - TCP flags
Targets

- Define what to do with the packet at this time
  - ACCEPT/DROP
  - QUEUE for user-space application
  - LOG any packet that matches
  - REJECT drops and returns error packet
  - RETURN enables packet to return to previous chain
  - <user-specified> passes packet to that chain
Examples

iptables -A INPUT -s 200.200.200.2 -j ACCEPT
iptables -A INPUT -s 200.200.200.1 -j DROP
iptables -A INPUT -s 200.200.200.1 -p tcp -j DROP
iptables -A INPUT -s 200.200.200.1 -p tcp --dport telnet -j DROP
iptables -A INPUT -p tcp --destination-port telnet -i ppp0 -j DROP