CSE543 - Computer and Network Security

Module: Firewalls

Professor Trent Jaeger
Problem

• All network flows were possible
  ‣ Into or out of our network
  ‣ To/from individual hosts and their processes
  ‣ We need to control access to protect confidentiality, integrity and secrecy
• What mechanism do we need?
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)
• **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

• These is useful for IP filtering, SPAM mitigation, …

• Q: What access control policies do these represent?
Stateful, Proxy, and Transparent

• Single packet may not contain sufficient 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/driving 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 …
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></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>
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 iptable rule set is evaluated at each
  - “PREROUTING”: before routing
  - “INPUT”: inbound to local destination
  - “FORWARD”: inbound but routed off host
  - “OUTPUT”: outbound to remote destination
  - “POSTROUTING”: after routing

![Diagram of Netfilter hooks](image)
The `iptables` firewall looks in the firewall table to seek if a rule in the chain associated with the current hook matches a packet, and executes the rule’s target if it does.

- **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
- **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

- **plist** is a rule spec
iptables Rule Parameters

- Things you can match on
  - Destination/Source
    - IP address range and netmask
  - Protocol of packet
    - ICMP, TCP, etc
  - Incoming/outgoing interface
  - Fragmented only
  - Target on rule match
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
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
Deep Packet Inspection

- **Deep packet inspection** looks into the internals of a packet to look for some application/content context
  - e.g., inspect HTTP for URLs that point to malicious websites
  - Can have serious privacy issues if done by, say COMCAST

- To specify a match in `iptables`
  - `iptables -A INPUT -p tcp -m string --algo bm --string ‘exe’`
    - matches to packet with content containing ‘exe’
  - `iptables -A INPUT -p tcp -m length --length 10:100`
    - matches to packet with length between 10 and 100 bytes
    - Also, can specify ‘greater than 10’ by `10:`
Firewall Policy Design

• So, what is the problem with the firewall rules...

```plaintext
accept tcp 192.168.0.0/16 any
deny tcp 192.168.1.0/24 any 3127
```

• This may be a simple problem, but

• Rules now have complex actions
  ‣ What kind of rules can we write in iptables?

• Including **stateful** rules that we did not discuss
• Static analysis tool for detecting incorrect, inefficient, or inconsistent firewall rules
  ‣ Using something called binary decision diagrams
• Finds real misconfigurations
  ‣ Classify misconfigurations
  ‣ Applies intra-, inter-, and across firewalls
Misconfigurations

• Violations
  ‣ What is the security goal?

• Inconsistencies (possibly between firewalls)
  ‣ Shadowing: Accepts (denies) packets already denied (accepted)
  ‣ Generalization: Matches superset of preceding, but takes a different action
  ‣ Correlation: Matches subset of preceding, but takes a different action

• Inefficiencies
  ‣ Redundancy: Remove rule and no change
  ‣ Verbosity: Summarize with fewer rules
Analysis

• What is static analysis?
  ‣ Analyze without running program (firewall rules)
  ‣ Use code, approximate all possible executions at once
    • E.g., track all packets that have been accepted (A), denied (D),
      diverted (F) at each rule - remaining (R) is implied

• Firewall rules are translated into a rule graph
  ‣ Linear for one rule sequence
  ‣ Firewalls form a tree relative to target

• Problems detected by comparing sets (A, D, F, R, P)
  ‣ (P_j, Deny) where P_j subset A_j - shadowing
  ‣ (P_j, Deny) where P_j intersect A_j = NULL - redundant

• Implemented as a BDD
Take Away

• A **firewall** is an authorization mechanism for network flows
  ‣ Control packet flows to subnets, hosts, ports
  ‣ Scan a rulebase for matching rule for packet
    • Like Windows ACLs, but with default accept

• We examined the Linux **iptables** firewall
  ‣ Netfilter hooks provide complete mediation
  ‣ Rule chains can be connected like subroutines

• However, firewall rules **may be misconfigured**
  ‣ FIREMAN detects violations, inconsistencies, and inefficiencies using static analysis of rule bases
    • Compare sets of packets at rule with those accepted, denied, etc.