Firewalls

Chien-Chung Shen

cshen@cis.udel.edu
The Need for Firewalls

• Internet connectivity is essential
  - however it creates a threat

• vs. host-based security services (e.g., intrusion detection), not cost-effective

• Inserted between premises network and Internet to establish a controlled link
  - can be a single computer system or a set of two or more systems working together

• Used as a perimeter defense
  - single choke point to impose security and auditing
  - insulates internal systems from external networks
Design Goals

• all traffic from inside to outside, and vice versa, must pass through firewall
• only authorized traffic as defined by the local security policy will be allowed to pass
• firewall itself is immune to penetration (hardened system with secured OS)
Firewall Capabilities and Limits

• **Capabilities**
  - defines a single choke point to simplify security management
  - provides a location for monitoring security events (e.g., audit)
  - convenient platform for several Internet functions that are not security related (e.g., NAT, logging)
  - can serve as the platform for IPSec and Virtual Private Network (VPN)

• **Limitations**
  - cannot protect against attacks bypassing firewall
  - may not protect fully against internal threats
  - improperly secured wireless LAN can be accessed from outside the organization
  - laptop, PDA, or portable storage device may be infected outside the corporate network then used internally
Firewall Architecture (1)

- Firewalls can be designed to operate at any of the following three layers in TCP/IP protocol stack
  - **Transport** layer (e.g., packet filtering with `iptables`): examine every packet, check its IP header and its higher-level protocol headers (in order to figure out, say, whether it is TCP, UDP, ICMP packet, etc.) to decide whether or not to let the packet through and to determine whether or not to change any header fields
  - **Application** layer (e.g., HTTP proxy): examines requested session for whether it should be allowed or disallowed based on where the session request is coming from and the purpose of the requested session. Such firewalls are built with proxy servers
  - **Shim** layer: layer between Application layer and Transport layer (e.g., SOCKS proxy)
Firewall Architecture (2)

• For truly application layer firewalls, need a separate firewall for each type of service. *E.g.*, separate firewalls for HTTP, FTP, SMTP, *etc*. Such firewalls are basically access control declarations built into the applications themselves. Typically, network admin enters such declarations in server config files of applications.

• Shim layer traps the application-level calls from intranet clients for connection to the servers in the internet
  - a proxy server can monitor all session requests that are routed through it in an application-independent manner to check the requested sessions for their legitimacy
  - only the proxy server, serving as a firewall, would require direct connectivity to the internet and the rest of the intranet can “hide” behind the proxy server
Packet Filtering Firewall

• Take advantage of fact that direct support for TCP/IP is built into kernels of all major OSes now

• In Linux, packet filtering firewall is configured with `iptables` module which inserts and deletes rules from kernel's packet filtering table
  – ordinarily, rules created by the `iptables` command would be lost on reboot
  – make the rules permanent with commands `iptables-save` and `iptables-restore`

• The latest packet filtering framework in Linux is `nftables`, which was merged into the Linux kernel mainline on January 2014. `nftables` was developed to address the main shortcoming of `iptables`, where `iptables`' packet filtering code is much too protocol specific (IPv4 vs. IPv6 vs. ARP, etc.), resulting in code replication when firewall engines are created with `iptables`
Firewall on Ubuntu

• **Iptables** is a user-space application program that allows system administrator to configure tables provided by the Linux kernel firewall (implemented as different **Netfilter** modules) and the chains and rules it stores.

• **Netfilter** is a framework inside Linux kernel which offers flexibility for various networking-related operations to be implemented in form of customized handlers:
  - offers various options for packet filtering, network address translation, and port translation
  - these functions provide the functionality required for directing packets through a network, as well as for providing ability to prohibit packets from reaching sensitive locations within a computer network
Firewall on Ubuntu

• Installing Ubuntu on laptop automatically activates the `iptables` firewall but with **empty** packet filter table

```
$ iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

• show that `iptables` is on and running; every packet will be subject to the policy `ACCEPT`

```
$ iptables -F   // flush table
```
Linux iptables

• Supports 4 tables: filter, nat, mangle, and raw

• `iptables -L == iptables -L -t filter`

• `iptables -t filter -X`  
  // delete user-defined chains
Stop Pinging

- `iptables -A INPUT -p icmp --icmp-type echo-request -j DROP`
  - `-A INPUT`: append a new rule to the INPUT chain of the filter table
  - `--icmp-type echo-request`: what specific subtype of ICMP packets this rule applies to
  - `-j DROP`: action to be taken (drop packets)
  - This rule says to drop all incoming ICMP packets that are of type echo-request
Allow ssh and Nothing Else

- `iptables -A INPUT -p tcp --destination-port 22 -j ACCEPT`
- `iptables -A INPUT -j REJECT`

- `-A INPUT`: append a new rule to the INPUT chain of the filter table
- `-p tcp`: rule is applied to TCP packets
- `--destination-port`: port #
- `-j ACCEPT`: accept all such packets

`$ iptables -L` and `ping` // check output

- How about `-j DROP`?
  - ping
  - nothing back (vs. reject with error message)

`$ iptables -A INPUT -j DROP` // check output
Reject All Connection Requests

• Use mangle table
  ```
  iptables -t mangle -A PREROUTING -p tcp -m tcp --tcp-flags SYN NONE -j DROP
  iptables -t mangle -L  // check mangle table
  ```

• Try ssh
• Can you ping?
Five Tables

• Four tables: filter, nat, mangle, raw, security
• Each consists of chains of rules
• Each packet is subject to each of the rules in a table and the fate of the packet is decided by the first matching rule
• In most common use cases you will only use two of these: filter and nat. The other tables are aimed at complex configurations involving multiple routers and routing decisions
filter Table

- Contains at least three rule chains:
  - INPUT: for processing all incoming packets
  - OUTPUT: for processing all outgoing packets
  - FORWARD: for processing all packets being routed through the machine

- INPUT, OUTPUT, and FORWARD chains of filter table are also referred to as built-in chains since they cannot be deleted
nat Table

- nat table is consulted when packet that creates a new connection is encountered
- nat stands for Network Address Translation
- When machine acts as router, it would need to alter either source IP address in the packet passing through, or destination IP address, or both
- Consists of three built-in chains:
  - PREROUTING for altering packets as soon as they come in
  - OUTPUT for altering locally-generated packets before routing
  - POSTROUTING for altering packets as they are about to go out
Notes on Home Networks

• When your machine is connected to your home or small-business network and you are behind, say, a wireless router/access-point, you are in a Class C private network.

• The allowed address range for such networks is 10.0.0/24.

• When packet in private network is routed out to Internet at large, it is subject to NAT.

• Same things happens when packet from Internet at large is routed to your machine in private network; it is also subject to NAT, which would be the reverse of address translation carried out for outgoing packet.
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
mangle Table

- Used for specialized packet alteration
- Has five rule chains:
  - PREROUTING for altering incoming packets before a routing decision is made concerning the packet
  - OUTPUT for altering locally generated outgoing packets
  - INPUT for altering packets coming into (destining to) the machine itself
  - FORWARD for altering packets being routed through the machine
  - POSTROUTING for altering packets immediately after the routing decision
raw Table

• Used for configuring exceptions to connection tracking rules
  - specify a sequence of rules for connection tracking, but, at the same time, don’t want to expose a particular category of packets to those rules
  - for configuring packets so that they are exempt from connection tracking.

• When a raw table is present, it takes priority over all other tables
security Table

• used for Mandatory Access Control networking rules
Packet Processing in iptables

- **Tables** consist of *chains*, which are lists of rules which are followed in order
  - The (default) filter table contains three built-in chains: INPUT, OUTPUT and FORWARD, which are activated at different points of the packet filtering process
  - The nat table includes PREROUTING, POSTROUTING, and OUTPUT chains

- Packet filtering is based on **rules**, which are specified by multiple *matches* (conditions the packet must satisfy so that the rule can be applied), and one **target** (action taken when packet matches all conditions)

- The typical things a rule might match on are what interface the packet came in on (e.g., eth0 or eth1), what type of packet it is (ICMP, TCP, or UDP), or the destination port of the packet
Packet Processing by filter Table

• When packet comes in (say, through Ethernet NIC) kernel first looks at destination of packet (step labeled ‘routing’)
• If routing decision is that packet is intended for the machine in which packet is being processed, packet passes to INPUT chain
• If incoming packet is destined for another network interface on the machine, then packet goes to FORWARD chain. If accepted by FORWARD chain, packet is sent to the other interface
• If program running on computer wants to send packet out of the machine, packet must traverse through OUTPUT chain. If it is accepted by any of the rules, it is sent to whatever interface packet is intended for
• Each rule in a chain examines packet header. If condition part of rule matches packet header, action specified by rule is taken. Otherwise, packet moves on to next rule
Packet Processing by filter Table

Routing decision

INPUT Chain rules

FORWARD Chain rules

OUTPUT Chain rules
Check Status of iptables

- Execute as root `lsmod | grep ip` (show the status of modules in Linux kernel)
- Or `iptables -L` for filter table (iptables -t nat -L)

Chain INPUT (*policy ACCEPT*)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
</table>

Chain FORWARD (*policy ACCEPT*)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
</table>

Chain OUTPUT (*policy ACCEPT*)

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
</table>

- Note **policy** shown for each built-in chain right next to name of chain
  - **policy** is what is applied to packet if it is **not** trapped by any rules in a chain
Firewall Scenario

- Allow for unrestricted Internet access from all the machines in LAN
- Allow for SSH access to the firewall machine from outside LAN
- Permit Auth, that is used by services like SMTP and IRC
- LAN is hosting a web server (on behalf of the whole LAN) and that this HTTPD server is running on machine 192.168.1.100. So the firewall must use NAT to redirect the incoming TCP port 80 requests to 192.168.1.100
- Accept ICMP Echo requests coming from outside
- Want firewall to respond back with TCP RST or ICMP Unreachable for incoming requests for blocked ports
- Firewall must log filter statistics on the external interface of the firewall machine
Solution (1)

```bash
#!/bin/sh

# macro for external interface:
ext_if = "eth0"
# macro for internal interface:
int_if = "eth1"

tcp_services = "22,113"
icmp_types = "ping"
comp_httpd = "192.168.1.100"

# NAT/Redirect
modprobe ip_nat_ftp
iptables -t nat -A POSTROUTING -o $ext_if -j MASQUERADE
iptables -t nat -i -A PREROUTING $ext_if -p tcp --dport 80 \
   -j DNAT --to-destination $comp_httpd

# filter table rules
# Forward only from external to webserver:
ip tables -A FORWARD -m state --state=ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i $ext_if -p tcp -d $comp_httpd --dport 80 --syn -j ACCEPT

# From internal is fine, rest rejected
iptables -A FORWARD -i $int_if -j ACCEPT
iptables -A FORWARD -j REJECT
```

IP masquerade: one type of NAT that allows all hosts on a private network to use the Internet at the price of a single IP address.
Solution (2)

# External can only come in to $tcp_services and $icmp_types
iptables -A INPUT -m state --state=ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i $ext_if -p tcp --dport $tcp_services --syn -j \ 
       ACCEPT
for icmp in $icmp_types; do
    iptables -A INPUT -p $icmp --icmp-type $icmp -j ACCEPT
done

# Internal and loopback are allowed to send anything:
iptables -A INPUT -i $int_if -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -j REJECT

# logging
echo "1" > /proc/sys/net/ipv4/ip_forward