Network Security

Routing and Firewalls

Radboud University Nijmegen, The Netherlands

Autumn 2014

A short recap

- IP spoofing by itself is easy
- Typically used in conjunction with other attacks, e.g.:
  - DOS attacks (e.g., SYN flooding)
  - TCP session stealing
  - Ping of death (classic, could say: historic)

A short recap

- IP spoofing by itself is easy
- Typically used in conjunction with other attacks, e.g.:
  - DOS attacks (e.g., SYN flooding)
  - TCP session stealing
  - Ping of death (classic, could say: historic)
- TCP session stealing needs ISN guessing (quite hard today)

A short recap

- IP spoofing by itself is easy
- Typically used in conjunction with other attacks, e.g.:
  - DOS attacks (e.g., SYN flooding)
  - TCP session stealing
  - Ping of death (classic, could say: historic)
- TCP session stealing needs ISN guessing (quite hard today)
- Standard tool for "raw" access to an open port: `netcat`
- For SSL connections use `openssl s_client`

A short recap

- IP spoofing by itself is easy
- Typically used in conjunction with other attacks, e.g.:
  - DOS attacks (e.g., SYN flooding)
  - TCP session stealing
  - Ping of death (classic, could say: historic)
- TCP session stealing needs ISN guessing (quite hard today)
- Standard tool for “raw” access to an open port: `netcat`
- For SSL connections use `openssl s_client`
- Discovering services on the network: portscan (`nmap`)
- Discovers open ports
- Various different approaches to (stealthy) scanning
- Can also fingerprint the OS of the target

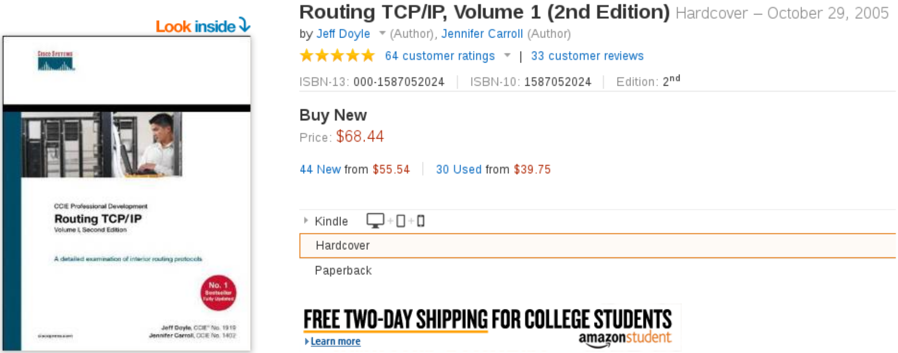
A short recap

- IP spoofing by itself is easy
- Typically used in conjunction with other attacks, e.g.:
  - DOS attacks (e.g., SYN flooding)
  - TCP session stealing
  - Ping of death (classic, could say: historic)
- TCP session stealing needs ISN guessing (quite hard today)
- Standard tool for "raw" access to an open port: `netcat`
- For SSL connections use `openssl s_client`
- Discovering services on the network: portscan (`nmap`)
- Discovers open ports
- Various different approaches to (stealthy) scanning
- Can also fingerprint the OS of the target
- Portknocking can hide open ports from scanner
- Various approaches, most recent one: TCP Stealth

Routing

- IP is responsible for delivering packets from one host to another host
- *Routing* is the process of finding a path to the destination
- Routers are (specialized) computers that forward packets between networks
- Routing is a very extensive and complex topic

Routing



Look inside
CCIE Professional Development
Routing TCP/IP
Volume I, Second Edition
A detailed examination of interior routing protocols
No. 1 Bestseller
Jeff Doyle, CCIE No. 1919
Jennifer Carroll, CCIE No. 1402

Routing TCP/IP, Volume 1 (2nd Edition) Hardcover – October 29, 2005

by Jeff Doyle (Author), Jennifer Carroll (Author)

★★★★★ 64 customer ratings | 33 customer reviews

ISBN-13: 000-1587052024 | ISBN-10: 1587052024 | Edition: 2nd

Buy New

Price: $68.44

44 New from $55.54 | 30 Used from $39.75

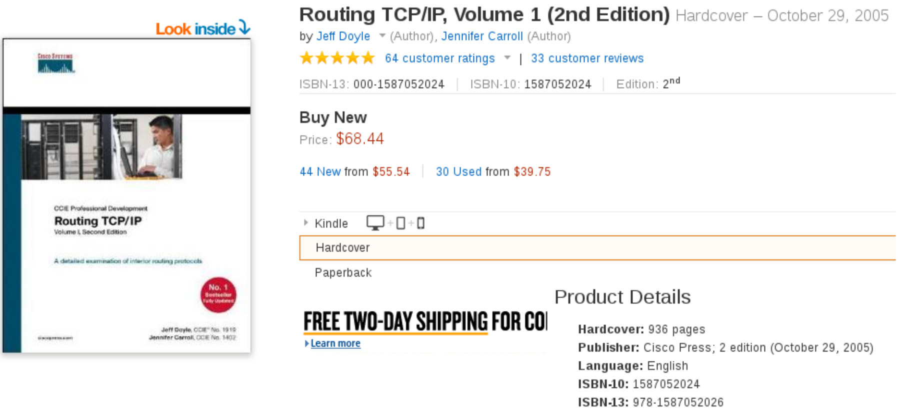
Kindle

Hardcover

Paperback

Source: http://www.amazon.com/Routing-TCP-IP-1-2nd/dp/1587052024/

Routing


Look inside
CCIE Professional Development
Routing TCP/IP
Volume I, Second Edition
A detailed examination of interior routing protocols
No. 1 Bestseller
Jeff Doyle, CCIE No. 1919
Jennifer Carroll, CCIE No. 1402

Routing TCP/IP, Volume 1 (2nd Edition) Hardcover – October 29, 2005

by Jeff Doyle (Author), Jennifer Carroll (Author)

★★★★★ 64 customer ratings | 33 customer reviews

ISBN-13: 000-1587052024 | ISBN-10: 1587052024 | Edition: 2nd

Buy New

Price: $68.44

44 New from $55.54 | 30 Used from $39.75

Kindle

Hardcover

Paperback

FREE TWO-DAY SHIPPING FOR CO

Learn more

Product Details

Hardcover: 936 pages

Publisher: Cisco Press; 2 edition (October 29, 2005)

Language: English

ISBN-10: 1587052024

ISBN-13: 978-1587052026

Source: http://www.amazon.com/Routing-TCP-IP-1-2nd/dp/1587052024/

traceroute

- Find out what route is used to, e.g., `www.google.com`:

```
traceroute www.google.com
```

`traceroute`

- Find out what route is used to, e.g., `www.google.com`:

```
traceroute www.google.com
```

- IP header has a *time-to-live* (TTL) field
- Each hop of the packet decreases the TTL by 1
- When TTL has reached zero, send back *ICMP time exceeded*

traceroute

- Find out what route is used to, e.g., `www.google.com`:

```
traceroute www.google.com
```

- IP header has a *time-to-live* (TTL) field
- Each hop of the packet decreases the TTL by 1
- When TTL has reached zero, send back *ICMP time exceeded*
- `traceroute` sends packets with increasing TTL:
    - Find first node: TTL=1
    - Find second node: TTL=2
    - . . .

traceroute

- Find out what route is used to, e.g., `www.google.com`:

```
traceroute www.google.com
```

- IP header has a *time-to-live* (TTL) field
- Each hop of the packet decreases the TTL by 1
- When TTL has reached zero, send back *ICMP time exceeded*
- `traceroute` sends packets with increasing TTL:
  - Find first node: TTL=1
  - Find second node: TTL=2
  - ...
- Can use UDP packets, ICMP echo requests (ping), or TCP SYN
- What really matters is only the TTL in the IP header

Routing on the Internet (highly simplified)

- The Internet is divided into multiple *Autonomous Systems (ASs)*
- Currently about $45,000$ Autonomous Systems

Routing on the Internet (highly simplified)

- The Internet is divided into multiple *Autonomous Systems (ASs)*
- Currently about $45,000$ Autonomous Systems
- Routing within one AS uses *Interior Gateway Protocols (IGPs)*

Routing on the Internet (highly simplified)

- The Internet is divided into multiple *Autonomous Systems (ASs)*
- Currently about $45,000$ Autonomous Systems
- Routing within one AS uses *Interior Gateway Protocols (IGPs)*
- Routing between ASs uses an *Exterior Gateway Protocol (EGP)*
- An AS is identified by its Autonomous System Number (ASN), managed by IANA

Routing on the Internet (highly simplified)

- The Internet is divided into multiple *Autonomous Systems (ASs)*
- Currently about $45,000$ Autonomous Systems
- Routing within one AS uses *Interior Gateway Protocols (IGPs)*
- Routing between ASs uses an *Exterior Gateway Protocol (EGP)*
- An AS is identified by its Autonomous System Number (ASN), managed by IANA
- Think of an AS as all networks under the control of one Internet Service Provider (ISP)

Routing on the Internet (highly simplified)

The classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASes.

Routing on the Internet (highly simplified)

> *The classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASes. Since this classic definition was developed, it has become common for a single AS to use several interior gateway protocols and sometimes several sets of metrics within an AS. The use of the term Autonomous System here stresses the fact that, even when multiple IGPs and metrics are used, the administration of an AS appears to other ASes to have a single coherent interior routing plan and presents a consistent picture of what networks are reachable through it.*
>
> —*RFC 1930*

Routing attacks

- Changing routes enables three kinds of attacks:
  - Detaching a target from the network (DOS)
  - Flooding a target with requests (DOS)
  - Becoming MitM

Static routing

- Simplest form of routing: manage all routes by hand (static routing)
- Linux supports multiple routing tables
- Most important routing table: `main`
- Show current routes with

  ```
  ip route show
  ```

Static routing

- Simplest form of routing: manage all routes by hand (static routing)
- Linux supports multiple routing tables
- Most important routing table: `main`
- Show current routes with

  ```
  ip route show
  ```

- Add route with `ip route add`, e.g.:

  ```
  ip route add 10.38.0.0/16 via 192.168.42.5
  ```

Static routing

- Simplest form of routing: manage all routes by hand (static routing)
- Linux supports multiple routing tables
- Most important routing table: `main`
- Show current routes with

  ```
  ip route show
  ```

- Add route with `ip route add`, e.g.:

  ```
  ip route add 10.38.0.0/16 via 192.168.42.5
  ```

- Most important use of static routes: set a default gateway:

  ```
  ip route add default via 192.168.42.1
  ```

Dynamic routing

- Static routing is often not enough:
  - Large risk of human error
  - Complex to configure for many/large networks
  - Cannot react to changes in the network

Dynamic routing

- Static routing is often not enough:
  - Large risk of human error
  - Complex to configure for many/large networks
  - Cannot react to changes in the network
- Alternative: *Dynamic (or adaptive) routing*
- Routers communicate information to their neighbors
- Build a table of efficient routes dynamically from this information

Dynamic routing

- Static routing is often not enough:
  - Large risk of human error
  - Complex to configure for many/large networks
  - Cannot react to changes in the network
- Alternative: *Dynamic (or adaptive) routing*
- Routers communicate information to their neighbors
- Build a table of efficient routes dynamically from this information
- Can combine static and dynamic routing
- Example: use dynamic routing, but configure one static default route (as backup)

RIP, OSPF, and IS-IS

Routing Information Protocol

- RIP is the traditional routing protocol of the Internet (RFC 1058 from 1988)
- Uses hop-count as metric (max hop-count: 15)
- Control messages on UDP, port 520
- RIPv2 introduced in 1993, latest RFC from 1998: RFC 2453
- Originally easily vulnerable to attacks (no authentication)
- MD5 authentication added in 1997 in RFC 2082

RIP, OSPF, and IS-IS

Routing Information Protocol

- RIP is the traditional routing protocol of the Internet (RFC 1058 from 1988)
- Uses hop-count as metric (max hop-count: 15)
- Control messages on UDP, port 520
- RIPv2 introduced in 1993, latest RFC from 1998: RFC 2453
- Originally easily vulnerable to attacks (no authentication)
- MD5 authentication added in 1997 in RFC 2082

Open Shortest Path First

- Very commonly used in corporate Networks
- Uses IP (protocol number 89)
- Supports authentication

RIP, OSPF, and IS-IS

Routing Information Protocol

- RIP is the traditional routing protocol of the Internet (RFC 1058 from 1988)
- Uses hop-count as metric (max hop-count: 15)
- Control messages on UDP, port 520
- RIPv2 introduced in 1993, latest RFC from 1998: RFC 2453
- Originally easily vulnerable to attacks (no authentication)
- MD5 authentication added in 1997 in RFC 2082

Intermediate System to Intermediate System

- De facto standard for ISPs
- Control messages on link layer
- Supports authentication

BGP

- Routing between different ASs uses the *Border Gateway Protocol*
- The BGP is thus an EGP

BGP

- Routing between different ASs uses the *Border Gateway Protocol*
- The BGP is thus an EGP
- Can also be used as an IGP (then called iBGP)

BGP

- Routing between different ASs uses the *Border Gateway Protocol*
- The BGP is thus an EGP
- Can also be used as an IGP (then called iBGP)
- Uses messages over TCP port 179
- Slightly more than $500,000$ active routing-table entries (AS6447)

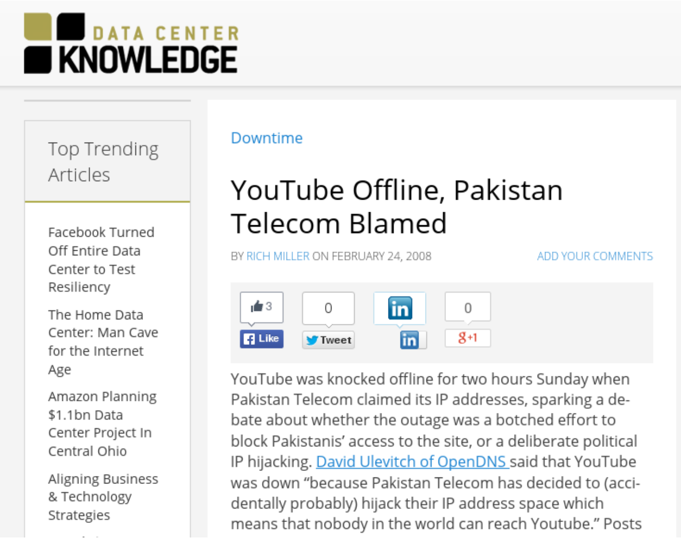
BGP

- Routing between different ASs uses the *Border Gateway Protocol*
- The BGP is thus an EGP
- Can also be used as an IGP (then called iBGP)
- Uses messages over TCP port 179
- Slightly more than $500,000$ active routing-table entries (AS6447)
- BGP security vulnerabilities have their own RFC (RFC 4272)

BGP

- Routing between different ASs uses the *Border Gateway Protocol*
- The BGP is thus an EGP
- Can also be used as an IGP (then called iBGP)
- Uses messages over TCP port 179
- Slightly more than $500,000$ active routing-table entries (AS6447)
- BGP security vulnerabilities have their own RFC (RFC 4272)
- BGP routing can be political, see "Schengen routing"

Pakistan knocks Youtube offline



DATA CENTER KNOWLEDGE

Top Trending Articles

- Facebook Turned Off Entire Data Center to Test Resiliency
- The Home Data Center: Man Cave for the Internet Age
- Amazon Planning $1.1bn Data Center Project In Central Ohio
- Aligning Business & Technology Strategies

Downtime

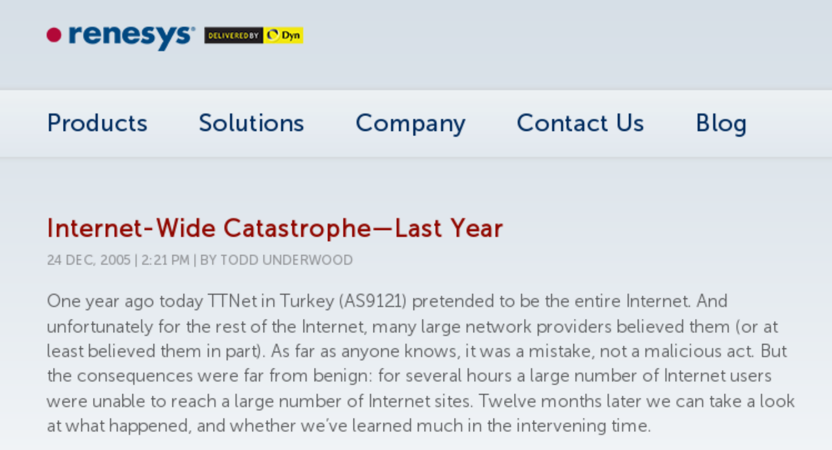
YouTube Offline, Pakistan Telecom Blamed

BY RICH MILLER ON FEBRUARY 24, 2008 ADD YOUR COMMENTS

YouTube was knocked offline for two hours Sunday when Pakistan Telecom claimed its IP addresses, sparking a debate about whether the outage was a botched effort to block Pakistanis' access to the site, or a deliberate political IP hijacking. David Ulevitch of OpenDNS said that YouTube was down "because Pakistan Telecom has decided to (accidentally probably) hijack their IP address space which means that nobody in the world can reach Youtube." Posts

Source: http://www.datacenterknowledge.com/archives/2008/02/24/youtube-offline-pakistan-telecom-blamed/

TTNet claims to be the Internet



Internet-Wide Catastrophe—Last Year

24 DEC, 2005 | 2:21 PM | BY TODD UNDERWOOD

One year ago today TTNet in Turkey (AS9121) pretended to be the entire Internet. And unfortunately for the rest of the Internet, many large network providers believed them (or at least believed them in part). As far as anyone knows, it was a mistake, not a malicious act. But the consequences were far from benign: for several hours a large number of Internet users were unable to reach a large number of Internet sites. Twelve months later we can take a look at what happened, and whether we've learned much in the intervening time.

Source: http://www.renesys.com/2005/12/internetwide-nearcatastrophela/

Source routing

- IP Header has SSRR and LSRR options
- SSRR (strict source and record route): Specify the complete routing path (go through only these hosts in exactly this order)
- LSRR (loose source and record route): Specify the a loose routing path (the specified hosts must be visited in the specified order)
- Idea in both cases: *The source specifies the route*
- Receiver reverts the route back to the target for the answer

Source routing

- IP Header has SSRR and LSRR options
- SSRR (strict source and record route): Specify the complete routing path (go through only these hosts in exactly this order)
- LSRR (loose source and record route): Specify the a loose routing path (the specified hosts must be visited in the specified order)
- Idea in both cases: *The source specifies the route*
- Receiver reverts the route back to the target for the answer

Source routing is evil

- Imagine that `joffrey` wants to IP spoof the address of `arya`
- `joffrey` can use LSRR and put himself into the route
- Now, the IP spoofing is not blind anymore: `joffrey` gets all the answers

ICMP redirect

- Consider three hosts, `arya`, `tyrion`, and `hodor` in the same network
- `arya`'s route to www.google.com goes through `hodor`, then `tyrion`

ICMP redirect

- Consider three hosts, `arya`, `tyrion`, and `hodor` in the same network
- `arya`'s route to www.google.com goes through `hodor`, then `tyrion`
- More efficient: route directly through `tyrion`
- `hodor` can notice this and inform `arya` about this through *ICMP redirect*

ICMP redirect

- Consider three hosts, `arya`, `tyrion`, and `hodor` in the same network
- `arya`'s route to www.google.com goes through `hodor`, then `tyrion`
- More efficient: route directly through `tyrion`
- `hodor` can notice this and inform `arya` about this through *ICMP redirect*
- Attack scenario:
  - `joffrey` spoofs IP address of `hodor` in ICMP redirect
  - Tells `arya` to route through `joffrey`
  - Now `joffrey` is MitM between `arya` and www.google.com

ICMP redirect

- Consider three hosts, `arya`, `tyrion`, and `hodor` in the same network
- `arya`'s route to www.google.com goes through `hodor`, then `tyrion`
- More efficient: route directly through `tyrion`
- `hodor` can notice this and inform `arya` about this through *ICMP redirect*
- Attack scenario:
  - `joffrey` spoofs IP address of `hodor` in ICMP redirect
  - Tells `arya` to route through `joffrey`
  - Now `joffrey` is MitM between `arya` and www.google.com
- Some limitations of this attack:
  - ICMP redirects will only be accepted for a route to a recently contacted host
  - 10 minutes
  - `arya` needs to accept ICMP redirect, this is configured in `/proc/sys/net/ipv4/conf/*/accept_redirects`

DHCP

- Typical way to hand out IP addresses: Dynamic Host Configuration Protocol (DHCP)
- When entering a network, a computer asks for an IP (and other information)
- Sends DHCP discovery packets; DHCP server answers
- Client requests various information
- DHCP server answers, typically with a network configuration

DHCP

- Typical way to hand out IP addresses: Dynamic Host Configuration Protocol (DHCP)
- When entering a network, a computer asks for an IP (and other information)
- Sends DHCP discovery packets; DHCP server answers
- Client requests various information
- DHCP server answers, typically with a network configuration

Rogue DHCP

- Attacker can answer DHCP requests faster
- Knock clients offline by providing unroutable IP addresses
- More imporantly: communicate himself as *default gateway*
- Can become MitM between the requesting client and the outside

Firewalls

Definition

A *firewall* is a concept for separating networks, typically together with technical means to implement this concept.

Firewalls

Definition

A *firewall* is a concept for separating networks, typically together with technical means to implement this concept.

- Firewalls can separate networks on different levels
- Most common: packet filtering on the internet and transport layers
- Often combined with filters on application level
- Finally: There are filters on lower level (e.g., MAC filters)

“Personal Firewalls”

- Many software products called “Personal Firewall” or “Desktop Firewall”
- Intended to protect against certain attacks on a local machine
- Typical things those products do:
  - Block access to network ports
  - Allow/deny network access only to certain applications
  - Monitor network access of applications

“Personal Firewalls”

- Many software products called “Personal Firewall” or “Desktop Firewall”
- Intended to protect against certain attacks on a local machine
- Typical things those products do:
  - Block access to network ports
  - Allow/deny network access only to certain applications
  - Monitor network access of applications
- Central problem: Most users don’t have a concept

“Personal Firewalls”

- Many software products called “Personal Firewall” or “Desktop Firewall”
- Intended to protect against certain attacks on a local machine
- Typical things those products do:
  - Block access to network ports
  - Allow/deny network access only to certain applications
  - Monitor network access of applications
- Central problem: Most users don’t have a concept
- Questionable how useful the features are:
  - If I want a port closed, I don’t open it in the first place
  - Can typically use an allowed application (web browser) to send data out

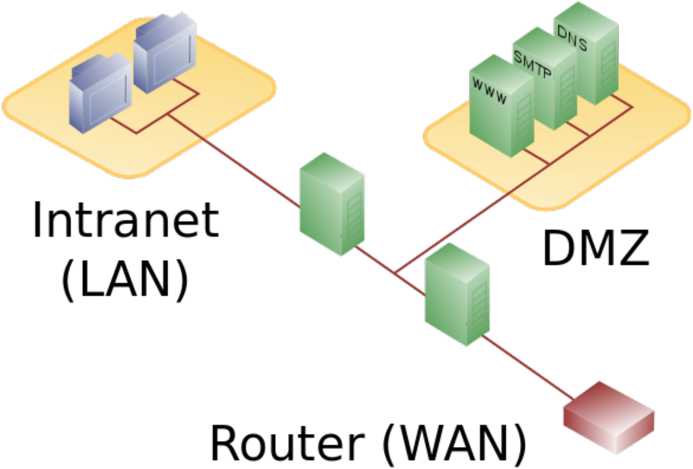
“Personal Firewalls”

- Many software products called “Personal Firewall” or “Desktop Firewall”
- Intended to protect against certain attacks on a local machine
- Typical things those products do:
  - Block access to network ports
  - Allow/deny network access only to certain applications
  - Monitor network access of applications
- Central problem: Most users don’t have a concept
- Questionable how useful the features are:
  - If I want a port closed, I don’t open it in the first place
  - Can typically use an allowed application (web browser) to send data out
- Potentially dangerous: additional piece of software with very highly privileged access!

Firewall layout and DMZs

- Common firewall layout separates three networks
  - The Internet
  - The Local Area Network
  - A de-militarized zone (DMZ)
- DMZ contains the servers that are accessible from the Internet

Firewall layout and DMZs



Source: http://en.wikipedia.org/wiki/DMZ_(computing)

iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules

iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*

`iptables`

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules

iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules
- Default table is `filter` with 3 chains: `INPUT`, `FORWARD`, and `OUTPUT`

iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules
- Default table is `filter` with 3 chains: `INPUT`, `FORWARD`, and `OUTPUT`
- Rules consist of *packet criteria* and a *target*
- Default targets:
  - A user-defined chain
  - One of `ACCEPT`, `DROP`, `RETURN`

iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules
- Default table is `filter` with 3 chains: `INPUT`, `FORWARD`, and `OUTPUT`
- Rules consist of *packet criteria* and a *target*
- Default targets:
  - A user-defined chain
  - One of `ACCEPT`, `DROP`, `RETURN`
- `RETURN` leaves the current chain and returns to calling chain

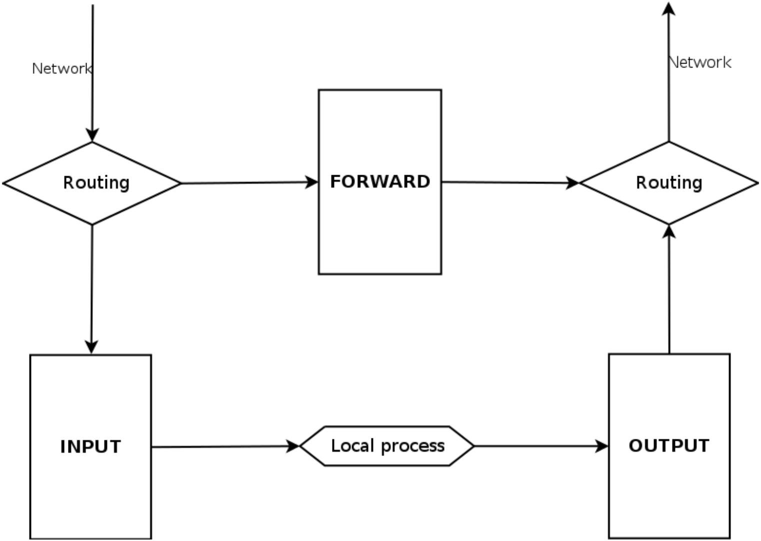
iptables

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules
- Default table is `filter` with 3 chains: `INPUT`, `FORWARD`, and `OUTPUT`
- Rules consist of *packet criteria* and a *target*
- Default targets:
  - A user-defined chain
  - One of `ACCEPT`, `DROP`, `RETURN`
- `RETURN` leaves the current chain and returns to calling chain
- Addionally helpful target: `REJECT`
- `--reject-with` specifies what error message to send (e.g., `icmp-port-unreachable` or `tcp-reject`)

`iptables`

- Linux kernel has powerful `netfilter` framework
- `iptables` is a tool to modify `netfilter` rules
- `iptables` defines multiple *tables*, each table with multiple *chains*, each chain with multiple *rules*
- Packets traverse the chains and are filtered and modified according to the rules
- Default table is `filter` with 3 chains: `INPUT`, `FORWARD`, and `OUTPUT`
- Rules consist of *packet criteria* and a *target*
- Default targets:
  - A user-defined chain
  - One of `ACCEPT`, `DROP`, `RETURN`
- `RETURN` leaves the current chain and returns to calling chain
- Addionally helpful target: `REJECT`
- `--reject-with` specifies what error message to send (e.g., `icmp-port-unreachable` or `tcp-reject`)
- Additional to rules, each of the 3 chains also has a *policy*
- The policy defines the default behavior (if no rule matches)

Packet processing with the `filter` table



Simple `iptables` examples

- Flush all tables:

  ```
  iptables -F
  ```

- Flush the `INPUT` chain

  ```
  iptables -F INPUT
  ```

Simple `iptables` examples

- Flush all tables:

  ```
  iptables -F
  ```

- Flush the `INPUT` chain

  ```
  iptables -F INPUT
  ```

- Set the `INPUT` policy to `DROP`:

  ```
  iptables -P INPUT DROP
  ```

Simple `iptables` examples

- Flush all tables:

  ```
  iptables -F
  ```

- Flush the `INPUT` chain

  ```
  iptables -F INPUT
  ```

- Set the `INPUT` policy to `DROP`:

  ```
  iptables -P INPUT DROP
  ```

- Allow ICMP echo request/reply (ping) from outside:

  ```
  iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
  iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
  ```

Simple iptables examples

- Flush all tables:

```
iptables -F
```

- Flush the INPUT chain

```
iptables -F INPUT
```

- Set the INPUT policy to DROP:

```
iptables -P INPUT DROP
```

- Allow ICMP echo request/reply (ping) from outside:

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

- Allow outbound DNS requests:

```
iptables -A OUTPUT -p udp -o eth0 --dport 53 -j ACCEPT
iptables -A INPUT -p udp -i eth0 --sport 53 -j ACCEPT
```

Stateful firewalls with `iptables`

- So far, the rules are stateless (don't know context)
- Most firewalls need stateful behaviour (in particular, for TCP):
  - I don't want external hosts to connect to port 12345
  - I do want external hosts to send data back to client port 12345

Stateful firewalls with iptables

- So far, the rules are stateless (don't know context)
- Most firewalls need stateful behaviour (in particular, for TCP):
  - I don't want external hosts to connect to port 12345
  - I do want external hosts to send data back to client port 12345
- iptables has multiple modules, use conntrack module for stateful firewall
- Example: Allow all incoming packets that belong to established or related connection:

```
iptables -A INPUT -m conntrack \
         --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Stateful firewalls with iptables

- So far, the rules are stateless (don't know context)
- Most firewalls need stateful behaviour (in particular, for TCP):
  - I don't want external hosts to connect to port 12345
  - I do want external hosts to send data back to client port 12345
- iptables has multiple modules, use conntrack module for stateful firewall
- Example: Allow all incoming packets that belong to established or related connection:

```
iptables -A INPUT -m conntrack \
         --ctstate RELATED,ESTABLISHED -j ACCEPT
```

- Most important connection states:
  - NEW: first packet of a connection
  - ESTABLISHED: Have seen packets of this connection before
  - RELATED: New connection, which is "related" to an ESTABLISHED connection

NAT

- IPv4 has 32-bit addresses, i.e., at most $4,294,967,296$ devices
- Very hard to use all addresses because of network separation
- We're running out of addresses!

NAT

- IPv4 has 32-bit addresses, i.e., at most $4,294,967,296$ devices
- Very hard to use all addresses because of network separation
- We're running out of addresses!
- Long-term solution: IPv6 (128-bit addresses)
- Short term work-around: Network Address Translation (NAT):
  - Multiple hosts in a local network (e.g., 192.168.0.0/16 or 10.0.0.0/8)
  - Only one host (the *gateway*) has an IP address routed in the Internet
  - "Shares" Internet connection to other hosts by rewriting the source IP address for outgoing packets
  - Remembers connection (IP+Port) to rewrite destination IP address on incoming packets

NAT

- IPv4 has 32-bit addresses, i.e., at most $4,294,967,296$ devices
- Very hard to use all addresses because of network separation
- We're running out of addresses!
- Long-term solution: IPv6 (128-bit addresses)
- Short term work-around: Network Address Translation (NAT):
    - Multiple hosts in a local network (e.g., 192.168.0.0/16 or 10.0.0.0/8)
    - Only one host (the *gateway*) has an IP address routed in the Internet
    - "Shares" Internet connection to other hosts by rewriting the source IP address for outgoing packets
    - Remembers connection (IP+Port) to rewrite destination IP address on incoming packets
- Strictly speaking, NAT is a more general concept
- This kind of NAT is also known as *IP Masquerading*

NAT example

- Three nodes in a local network:
  - `tyrion` 192.168.42.1
  - `arya` 192.168.42.2
  - `hodor` 192.168.42.3
- `tyrion` additionally has the (external) address 123.45.67.89
- `arya` and `hodor` use `tyrion` as default gateway

NAT example

- Three nodes in a local network:
  - `tyrion` 192.168.42.1
  - `arya` 192.168.42.2
  - `hodor` 192.168.42.3
- `tyrion` additionally has the (external) address 123.45.67.89
- `arya` and `hodor` use `tyrion` as default gateway
- `arya` connects (through `tyrion`) to www.google.com, Port 80, using source port 11111
- `hodor` connects (through `tyrion`) to www.google.com, Port 80, using source port 22222

NAT example

- Three nodes in a local network:
  - `tyrion` 192.168.42.1
  - `arya` 192.168.42.2
  - `hodor` 192.168.42.3
- `tyrion` additionally has the (external) address 123.45.67.89
- `arya` and `hodor` use `tyrion` as default gateway
- `arya` connects (through `tyrion`) to www.google.com, Port 80, using source port 11111
- `hodor` connects (through `tyrion`) to www.google.com, Port 80, using source port 22222
- `tyrion` rewrites source address for both connections to 123.45.67.89

NAT example

- Three nodes in a local network:
  - `tyrion` 192.168.42.1
  - `arya` 192.168.42.2
  - `hodor` 192.168.42.3
- `tyrion` additionally has the (external) address 123.45.67.89
- `arya` and `hodor` use `tyrion` as default gateway
- `arya` connects (through `tyrion`) to www.google.com, Port 80, using source port 11111
- `hodor` connects (through `tyrion`) to www.google.com, Port 80, using source port 22222
- `tyrion` rewrites source address for both connections to 123.45.67.89
- Incoming packets from `www.google.com` with dest. port 11111: Rewrite destination address to 192.168.42.2
- Incoming packets from `www.google.com` with dest. port 22222: Rewrite destination address to 192.168.42.3

NAT example

- Three nodes in a local network:
  - `tyrion` 192.168.42.1
  - `arya` 192.168.42.2
  - `hodor` 192.168.42.3
- `tyrion` additionally has the (external) address 123.45.67.89
- `arya` and `hodor` use `tyrion` as default gateway
- `arya` connects (through `tyrion`) to www.google.com, Port 80, using source port 11111
- `hodor` connects (through `tyrion`) to www.google.com, Port 80, using source port 22222
- `tyrion` rewrites source address for both connections to 123.45.67.89
- Incoming packets from `www.google.com` with dest. port 11111: Rewrite destination address to 192.168.42.2
- Incoming packets from `www.google.com` with dest. port 22222: Rewrite destination address to 192.168.42.3
- What happens if both `hodor` and `arya` connect to www.google.com with the *same* destination port?

NAT example

- Three nodes in a local network:
  - tyrion 192.168.42.1
  - arya 192.168.42.2
  - hodor 192.168.42.3
- tyrion additionally has the (external) address 123.45.67.89
- arya and hodor use tyrion as default gateway
- arya connects (through tyrion) to www.google.com, Port 80, using source port 11111
- hodor connects (through tyrion) to www.google.com, Port 80, using source port 22222
- tyrion rewrites source address for both connections to 123.45.67.89
- Incoming packets from www.google.com with dest. port 11111: Rewrite destination address to 192.168.42.2
- Incoming packets from www.google.com with dest. port 22222: Rewrite destination address to 192.168.42.3
- What happens if both hodor and arya connect to www.google.com with the *same* destination port?
- Answer: tyrion also rewrites the port

Some NAT remarks

NAT and ICMP

- NAT or IP masquerading relies on ports (UDP or TCP)
- ICMP messages do not have ports

Some NAT remarks

NAT and ICMP

- NAT or IP masquerading relies on ports (UDP or TCP)
- ICMP messages do not have ports
- For ICMP echo request/reply use the *Query ID* instead of the port
- FOR ICMP error messages it's more complex, but also possible
- This is specified in RFC 5508: NAT Behavioral Requirements for ICMP

Some NAT remarks

NAT and ICMP

- NAT or IP masquerading relies on ports (UDP or TCP)
- ICMP messages do not have ports
- For ICMP echo request/reply use the *Query ID* instead of the port
- FOR ICMP error messages it's more complex, but also possible
- This is specified in RFC 5508: NAT Behavioral Requirements for ICMP

Tethering

- Many (Android) phones offer sharing an Internet connection through *tethering*
- Tethering uses NAT (IP Masquerading)

Port forwarding

- So far, we can only establish connections from within the NAT network
- This is also known as “source-NAT”
- How about a server running inside a NAT network?

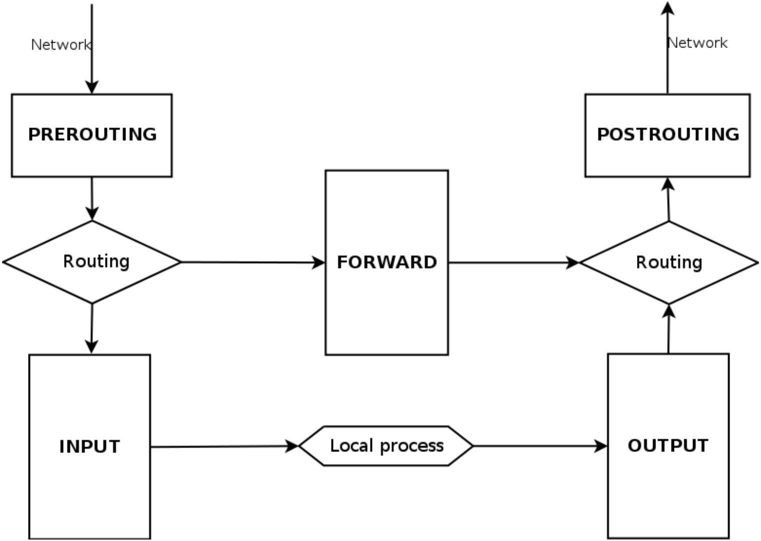
Port forwarding

- So far, we can only establish connections from within the NAT network
- This is also known as “source-NAT”
- How about a server running inside a NAT network?
- Can forward incoming connections to a server
- This is called *port forwarding* or *destination NAT*

NAT and port forwarding with iptables

- iptables has a nat table
- Three chains in this table: PREROUTING, POSTROUTING, and OUTPUT
- For now, only consider chains PREROUTING, and POSTROUTING

NAT and port forwarding with iptables



NAT and port forwarding with iptables

- Enabling NAT (IP Masquerading) through iptables:

  ```
  iptables -t nat -A POSTROUTING -j MASQUERADE
  ```

NAT and port forwarding with iptables

- Enabling NAT (IP Masquerading) through iptables:

  ```
  iptables -t nat -A POSTROUTING -j MASQUERADE
  ```

NAT and port forwarding with iptables

- Enabling NAT (IP Masquerading) through iptables:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

- Don't forget to enable IP forwarding:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

NAT and port forwarding with iptables

- Enabling NAT (IP Masquerading) through iptables:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

- Don't forget to enable IP forwarding:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Port forwarding from tyrion, port 1234 to arya, port 22:

```
iptables -A PREROUTING -t nat -p tcp \
         --dport 1234 -j DNAT --to 192.168.42.2:22
iptables -A FORWARD -p tcp -d 192.168.42.2 \
         --dport 22 -j ACCEPT
```

Tunneling

- `iptables` looks at traffic on the TCP/IP level
- `iptables` cannot distinguish between HTTP going to port 80 and SSH going to port 80
- Running an SSH server on ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) gets SSH through some firewalls

Tunneling

- `iptables` looks at traffic on the TCP/IP level
- `iptables` cannot distinguish between HTTP going to port 80 and SSH going to port 80
- Running an SSH server on ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) gets SSH through some firewalls
- Even better: SSH supports *tunneling*
- Tunneling generally means: place packets of one protocol into the payload of another protocol

Tunneling

- `iptables` looks at traffic on the TCP/IP level
- `iptables` cannot distinguish between HTTP going to port 80 and SSH going to port 80
- Running an SSH server on ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) gets SSH through some firewalls
- Even better: SSH supports *tunneling*
- Tunneling generally means: place packets of one protocol into the payload of another protocol
- SSH tunneling example:
  - You want to connect to SMTPS (port 465) server `mail.somedomain.com`
  - Port 465 is blocked but port 80 is open
  - You have an SSH server running on `mysshhost.nl`, port 80

Tunneling

- `iptables` looks at traffic on the TCP/IP level
- `iptables` cannot distinguish between HTTP going to port 80 and SSH going to port 80
- Running an SSH server on ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) gets SSH through some firewalls
- Even better: SSH supports *tunneling*
- Tunneling generally means: place packets of one protocol into the payload of another protocol
- SSH tunneling example:
    - You want to connect to SMTPS (port 465) server `mail.somedomain.com`
    - Port 465 is blocked but port 80 is open
    - You have an SSH server running on `mysshhost.nl`, port 80
    - Establish an SSH tunnel through

      ```
      ssh -p 80 -L 52428:mail.somedomain.com:465 mysshhost.nl
      ```

    - Connect to `localhost` at port 52428
    - SSH will forward the connection to `mail.somedomain.com`, port 465

Tunneling

- `iptables` looks at traffic on the TCP/IP level
- `iptables` cannot distinguish between HTTP going to port 80 and SSH going to port 80
- Running an SSH server on ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) gets SSH through some firewalls
- Even better: SSH supports *tunneling*
- Tunneling generally means: place packets of one protocol into the payload of another protocol
- SSH tunneling example:
  - You want to connect to SMTPS (port 465) server `mail.somedomain.com`
  - Port 465 is blocked but port 80 is open
  - You have an SSH server running on `mysshhost.nl`, port 80
  - Establish an SSH tunnel through

    ```
    ssh -p 80 -L 52428:mail.somedomain.com:465 mysshhost.nl
    ```

  - Connect to `localhost` at port 52428
  - SSH will forward the connection to `mail.somedomain.com`, port 465
  - To `mail.somedomain.com`, the connection looks like coming from `mysshhost.nl`

sshuttle

- Tunneling every connection separately is a hassle
- Often want to tunnel *all* traffic through SSH
- Extremely convenient tool: `sshuttle`
- Modify local firewall rules to tunnel all traffic through SSH:

  ```
  sshuttle --dns -vr mysshhost.nl 0/0
  ```

sshuttle

- Tunneling every connection separately is a hassle
- Often want to tunnel *all* traffic through SSH
- Extremely convenient tool: `sshuttle`
- Modify local firewall rules to tunnel all traffic through SSH:

  ```
  sshuttle --dns -vr mysshhost.nl 0/0
  ```

- Three main use cases for `sshuttle`:

sshuttle

- Tunneling every connection separately is a hassle
- Often want to tunnel *all* traffic through SSH
- Extremely convenient tool: `sshuttle`
- Modify local firewall rules to tunnel all traffic through SSH:

  ```
  sshuttle --dns -vr mysshhost.nl 0/0
  ```

- Three main use cases for `sshuttle`:
  - Tunnel everything through a firewall (once you have SSH access to somewhere)

sshuttle

- Tunneling every connection separately is a hassle
- Often want to tunnel *all* traffic through SSH
- Extremely convenient tool: `sshuttle`
- Modify local firewall rules to tunnel all traffic through SSH:

  ```
  sshuttle --dns -vr mysshhost.nl 0/0
  ```

- Three main use cases for `sshuttle`:
  - Tunnel everything through a firewall (once you have SSH access to somewhere)
  - Tunnel from an untrusted network to a (more) trusted network (VPN, later this lecture)

`sshuttle`

- Tunneling every connection separately is a hassle
- Often want to tunnel *all* traffic through SSH
- Extremely convenient tool: `sshuttle`
- Modify local firewall rules to tunnel all traffic through SSH:

```
sshuttle --dns -vr mysshhost.nl 0/0
```

- Three main use cases for `sshuttle`:
  - Tunnel everything through a firewall (once you have SSH access to somewhere)
  - Tunnel from an untrusted network to a (more) trusted network (VPN, later this lecture)
  - Circumvent country filters (e.g., watch a German stream of the worldcup in NL)
  - This last case needs SSH access to an unblocked country

Proxy Servers

- Additional to packet filtering on TCP/IP level: *proxy servers*
- A proxy server acts as an intermediary

Proxy Servers

- Additional to packet filtering on TCP/IP level: *proxy servers*
- A proxy server acts as an intermediary
- Two kinds of typical proxy servers:
  - Application-level proxy (understands high-level protocols, such as HTTP)
  - SOCKS proxy (for secure forwarding of TCP connections), can use SSH for this:

    ```
    ssh -fND localhost:8080 mysshhost.nl
    ```

Proxy Servers

- Additional to packet filtering on TCP/IP level: *proxy servers*
- A proxy server acts as an intermediary
- Two kinds of typical proxy servers:
  - Application-level proxy (understands high-level protocols, such as HTTP)
  - SOCKS proxy (for secure forwarding of TCP connections), can use SSH for this:

    ```
    ssh -fND localhost:8080 mysshhost.nl
    ```

- Similar to Proxy: *Application-level gateway (ALG)*
- Both (application-level) proxy and ALG can filter high-level protocols

Proxy Servers

- Additional to packet filtering on TCP/IP level: *proxy servers*
- A proxy server acts as an intermediary
- Two kinds of typical proxy servers:
  - Application-level proxy (understands high-level protocols, such as HTTP)
  - SOCKS proxy (for secure forwarding of TCP connections), can use SSH for this:

    ```
    ssh -fND localhost:8080 mysshhost.nl
    ```

- Similar to Proxy: *Application-level gateway (ALG)*
- Both (application-level) proxy and ALG can filter high-level protocols
- Can place proxies/ALGs in DMZ, then have no traffic go directly from the LAN to the Internet

Tunneling through an HTTP proxy

- Most common form of proxies: HTTP(S) proxies
- Fairly common restrictive firewall configuration:
  - Allow only HTTP(S) traffic (and DNS through UDP)
  - Allow HTTPS traffic only through local (filtering) proxy

Tunneling through an HTTP proxy

- Most common form of proxies: HTTP(S) proxies
- Fairly common restrictive firewall configuration:
  - Allow only HTTP(S) traffic (and DNS through UDP)
  - Allow HTTPS traffic only through local (filtering) proxy
- Can imagine an HTTP(S) server and client that tunnel SSH from HTTP payload
- Serious configuration effort

Tunneling through an HTTP proxy

- Most common form of proxies: HTTP(S) proxies
- Fairly common restrictive firewall configuration:
  - Allow only HTTP(S) traffic (and DNS through UDP)
  - Allow HTTPS traffic only through local (filtering) proxy
- Can imagine an HTTP(S) server and client that tunnel SSH from HTTP payload
- Serious configuration effort
- HTTP CONNECT() to the rescue: HTTP command for tunneling
- Very often allowed to support HTTPS

Tunneling through an HTTP proxy

- Most common form of proxies: HTTP(S) proxies
- Fairly common restrictive firewall configuration:
  - Allow only HTTP(S) traffic (and DNS through UDP)
  - Allow HTTPS traffic only through local (filtering) proxy
- Can imagine an HTTP(S) server and client that tunnel SSH from HTTP payload
- Serious configuration effort
- HTTP CONNECT() to the rescue: HTTP command for tunneling
- Very often allowed to support HTTPS
- Can use HTTP CONNECT() to tunnel SSH through an HTTP(S) proxy:

```
ssh user@server -o "ProxyCommand corkscrew \
                     PROXY_IP PROXY_PORT \
                     DESTINATION_IP DESTINATION_PORT"
```

Tunneling through an HTTP proxy

- Most common form of proxies: HTTP(S) proxies
- Fairly common restrictive firewall configuration:
  - Allow only HTTP(S) traffic (and DNS through UDP)
  - Allow HTTPS traffic only through local (filtering) proxy
- Can imagine an HTTP(S) server and client that tunnel SSH from HTTP payload
- Serious configuration effort
- HTTP CONNECT() to the rescue: HTTP command for tunneling
- Very often allowed to support HTTPS
- Can use HTTP CONNECT() to tunnel SSH through an HTTP(S) proxy:

```
ssh user@server -o "ProxyCommand corkscrew \
                     PROXY_IP PROXY_PORT \
                     DESTINATION_IP DESTINATION_PORT"
```

- Additional homework: `apt-get install sshuttle corkscrew` (some day you'll thank me ;-))