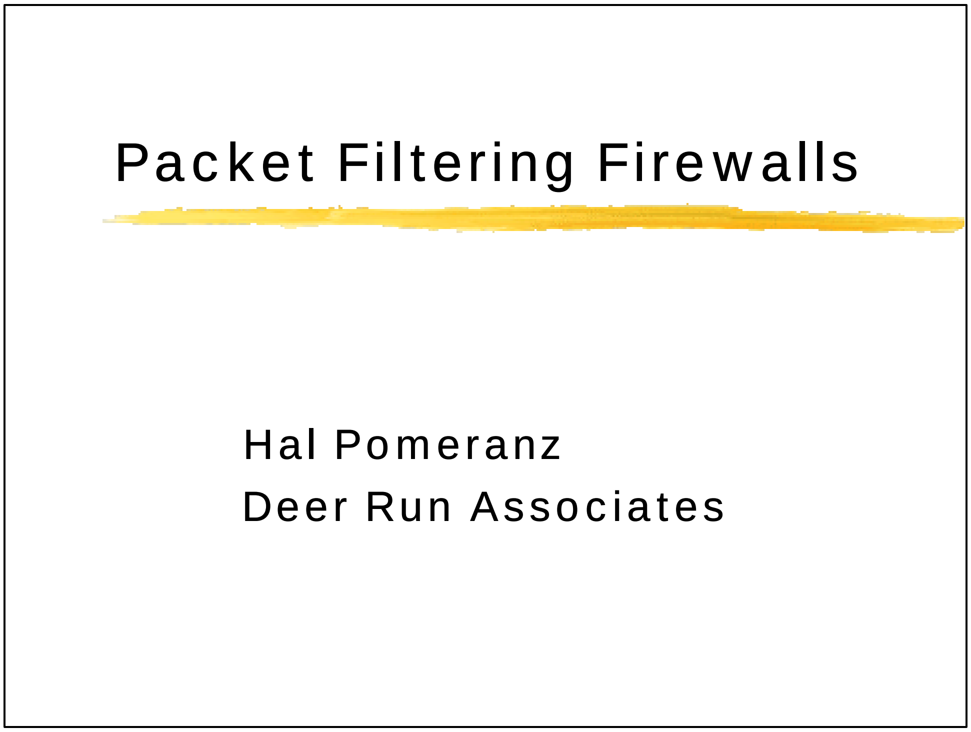
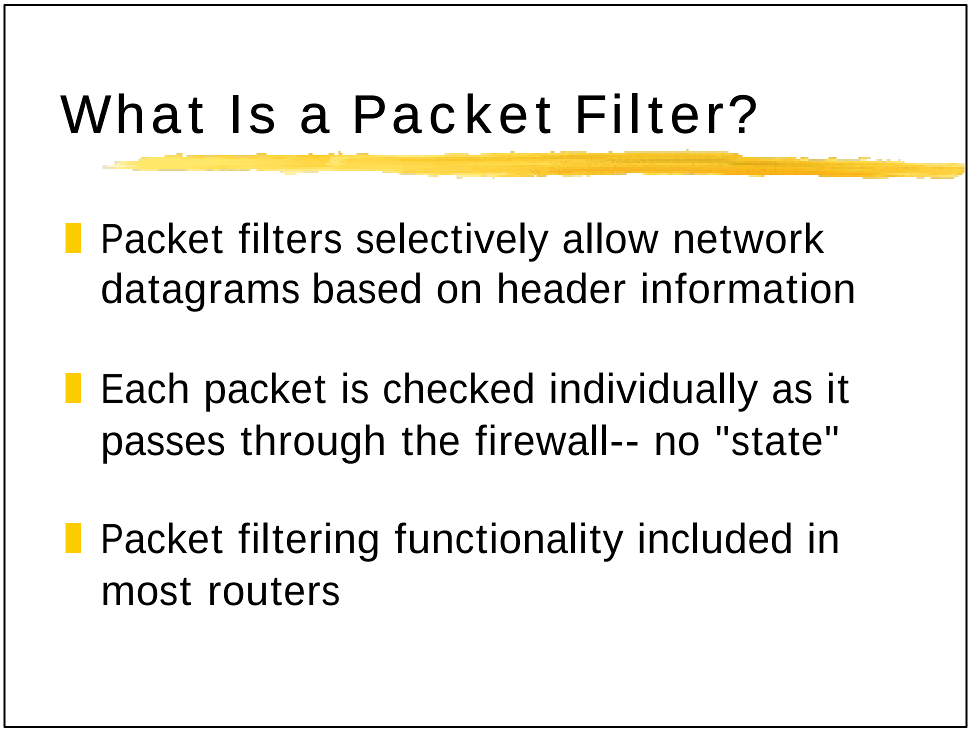


Packet Filtering Firewalls

Hal Pomeranz
Deer Run Associates

Hal Pomeranz * Founder/CEO * *hal@deer-run.com*
Deer Run Associates * 7145 Homewood Drive * Oakland, CA 94611
+1 510-339-7740 (voice and fax) * *http://www.deer-run.com/*

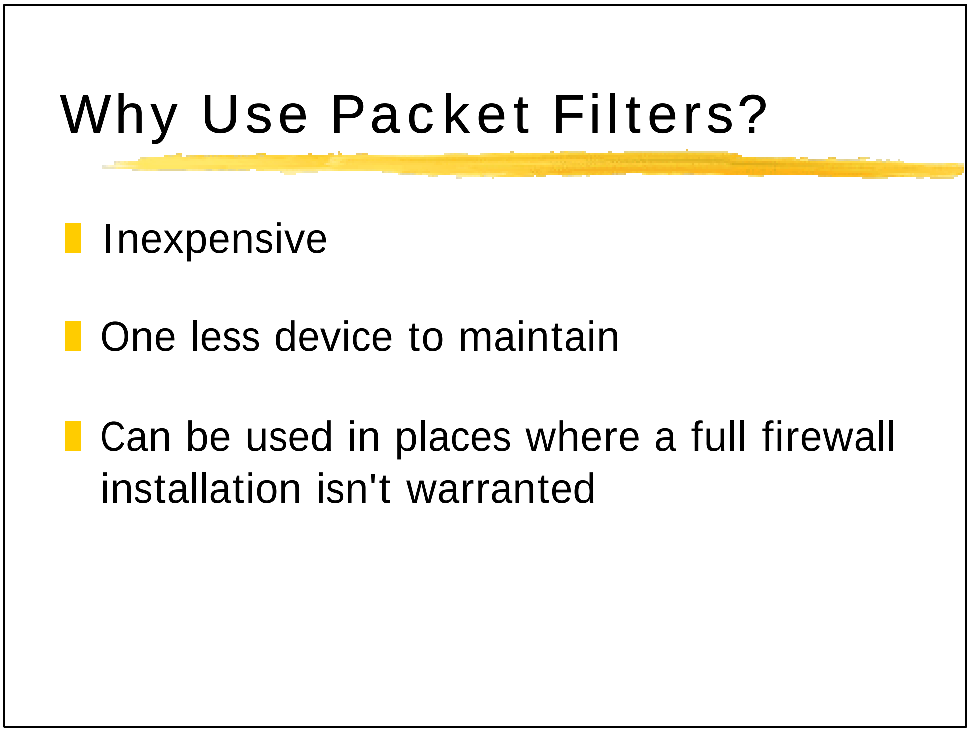


What Is a Packet Filter?

- Packet filters selectively allow network datagrams based on header information
- Each packet is checked individually as it passes through the firewall-- no "state"
- Packet filtering functionality included in most routers

Packet filtering functionality (built into most modern routers) allows the router to selectively permit or deny network traffic based on information in the network headers. Standard packet filters don't tend to read deeper than the header information for performance reasons-- unpacking arbitrary network datagrams would be difficult to impossible (encrypted traffic).

In general, each packet is considered individually without regard to any packets that may have been inspected before. As we will see later, this makes it difficult for packet filters to securely allow certain types of network traffic (notably FTP sessions). More modern "stateful" packet filters exist as commercial products from Checkpoint, Raptor, TIS, et al.

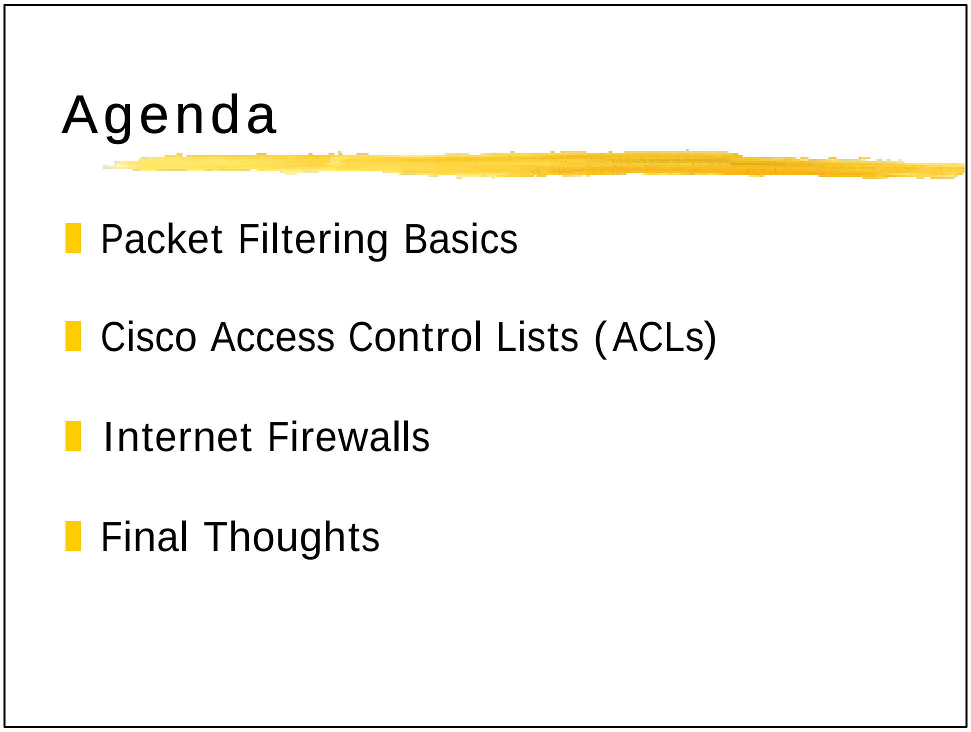


Why Use Packet Filters?

- Inexpensive
- One less device to maintain
- Can be used in places where a full firewall installation isn't warranted

Packet filters can be an inexpensive way for many organizations to provide basic firewalling capability for their networks. Since the security functionality is already incorporated in a device that the organization maintains, there should also be administrative savings.

In many cases, different parts of the same company may want to protect themselves from each other but not need a full-blown commercial firewall. For example, an organization might want to segregate its development lab networks from its production networks, or keep LAN protocols on separate networks from colliding. Applying packet filters at the router level is an inexpensive option for achieving such goals.



Agenda

- Packet Filtering Basics
- Cisco Access Control Lists (ACLs)
- Internet Firewalls
- Final Thoughts

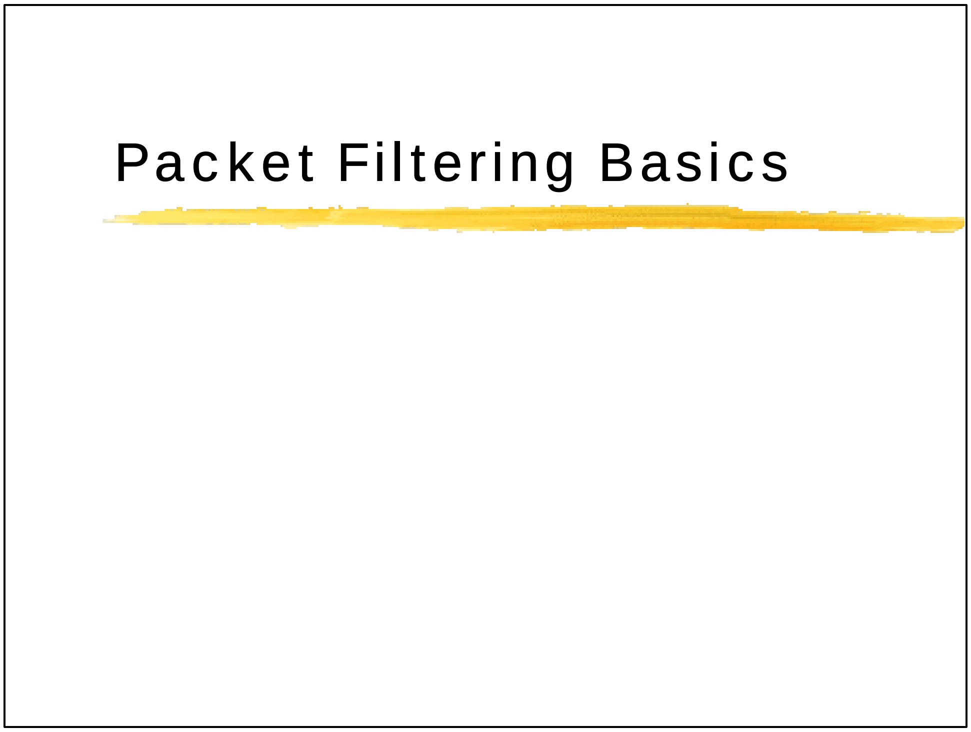
The rest of this presentation is divided into four sections:

Packet Filtering Basics-- How packet filters work, IP header information

Cisco Access Control Lists-- A high-speed introduction to the basics of writing packet filters for Cisco routers.

Internet Firewalls-- Creating a simple Internet firewall using Cisco ACLs

Final Thoughts-- Caveats and some additional information about internal firewalls and stateful firewalls



Packet Filtering Basics

Before you can begin writing firewalls you need to understand how packet filters work and the contents of IP datagram headers that packet filters operate on.

How Do Packet Filters Work?

- IP headers contain lots of information:
  - Protocol
  - Source and Destination Address
  - Source and Destination Port
  - Connection Status
- "Safe" traffic is identified using these parameters, all other packets are blocked

When packet filtering functionality is enabled, each packet passing through the router gets shunted to a special routine. This routine pulls out various pieces of information from the IP header of the packet, and then consults a list of rules configured by the administrator.

Each rule can specify any or all of the pieces of IP header information available to the router. If the parameters specified by the rule match the information in the packet header then that packet is permitted or denied as specified in the rule. Generally speaking, packet filters follow a "first match and exit" behavior-- the first rule to match the packet is the one that decides the destiny of that packet, so order is *very* important.

Packet filters tend to operate in a "default deny" stance-- any traffic that isn't specifically allowed by the filtering rules is automatically dropped. Combining wildcarding with a "permit" type rule allows the administration to configure a "default permit" stance if desired.

Some packet filters allow the administrator to specify what information the remote user gets back if a packet is dropped by the filter. In some cases you will want to send back an "administratively unreachable" message, sometimes you just want the packets to black-hole with no message so outsiders can't figure out what traffic you're blocking.

Protocol Types

ICMP	*Well-known user app is* ***`ping`*** *Useful but potentially dangerous msgs*
TCP	***`telnet`****, FTP, Web, etc.* *Easiest protocol to filter safely*
UDP	*Often used by video/audio apps* *Difficult to filter safely*

ICMP messages are used for low-level network control and testing. The `ping` program uses ICMP messages, and while this program is useful it can also be used by outsiders to map your internal network or wage a denial of service attack. Other ICMP messages can actually cause networked devices to re-route packets out a different gateway and otherwise play havoc with your network's configuration. Blocking many ICMP messages may be a good idea.

TCP-based services are the backbone of the Internet. Standard Internet clients like the `telnet` and `ftp` programs, as well as your Web browser use TCP. Email and USENET are shipped around using TCP. TCP-based programs are easy to filter because TCP connections (unlike ICMP and UDP streams) have a notion of connection state which can be used to create "one-way" filters.

UDP is used when a more lightweight protocol is needed-- typically video and audio streaming apps. The difficulty is that UDP has no sense of connection state-- it's difficult for a packet filter to determine whether a given UDP datagram is part of a session already in progress, or the start of a new (and potentially malicious) session.

Well-Known Ports

- Each end of a network connection is *bound* to a specific port
- Common servers assigned to well-known ports:

SMTP (email):	`25/tcp`
telnet:	`23/tcp`
HTTP (Web):	`80/tcp`

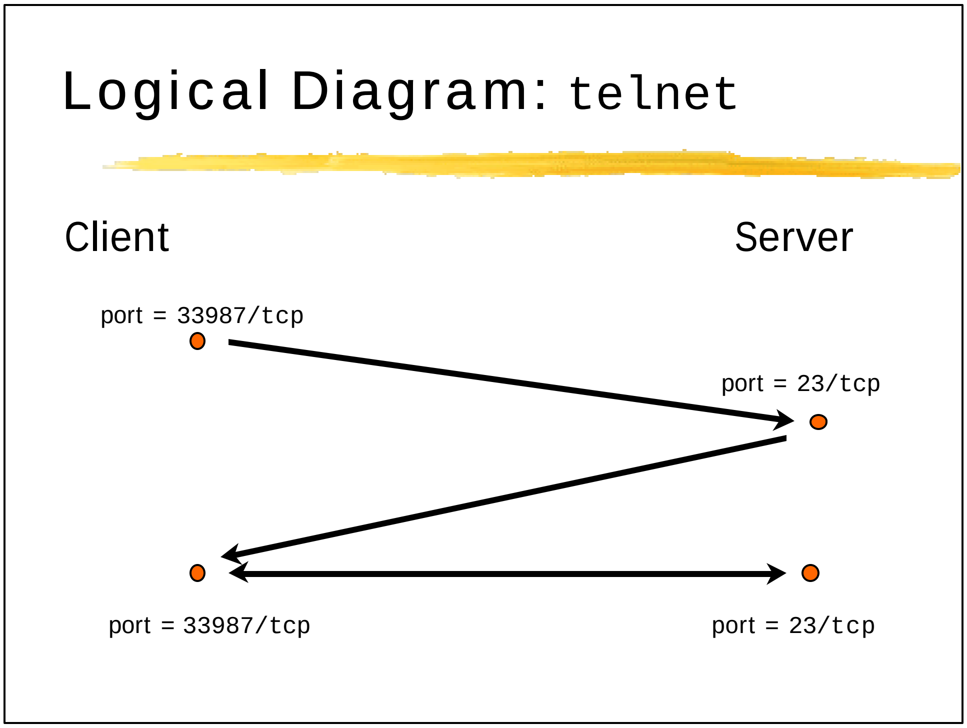
- Client side chooses a random port number

Two machines communicate across a network by directing packets of information at each other using unique IP addresses. However, any given machine may have dozens of simultaneous network connections happening at any given moment. A *network port* is a logical construct which allows a machine to keep multiple connections separate.

There are 16 bits of network port numbers-- TCP ports are distinct from UDP ports. Ports 0 through 1023 are reserved and not generally available to normal users. Furthermore, RFC1700 documents certain “well-known” ports that have been assigned to common network servers.

Well-known ports allow client software, e.g. the `telnet` program, to easily contact the appropriate server at the remote host. It would be impractical for a client to randomly probe the remote machine trying to find a server that will talk to it.

However, since the well-known ports constrain certain servers to listen on a fixed port number, an administrator can prevent access to a certain service by stopping packets targeted at a given network port. This is where packet filters come in.



Logical Diagram: `telnet`

Client

Server

port = `33987/tcp`

port = `23/tcp`

port = `33987/tcp`

port = `23/tcp`

In general, network clients grab a random unused port above 1023 (remember, ports lower than this are reserved). The client then constructs an initial packet with this random port number and its own IP address in the *source* portion of the packet. The *destination* is the IP address of the remote server and the well-known port appropriate for the given service. In this case, we're looking at a `telnet` client-- the well-known port for this service is `23/tcp`.

The server sends back an acknowledgement packet using the source IP address and port from the initial packet. Remember in this case the server uses its own IP address and port `23/tcp` in as the *source address* and the IP address and random port selected by the client are in the *destination* fields.

The client now acknowledges the server's acknowledgement and the session proceeds.

Connection Status

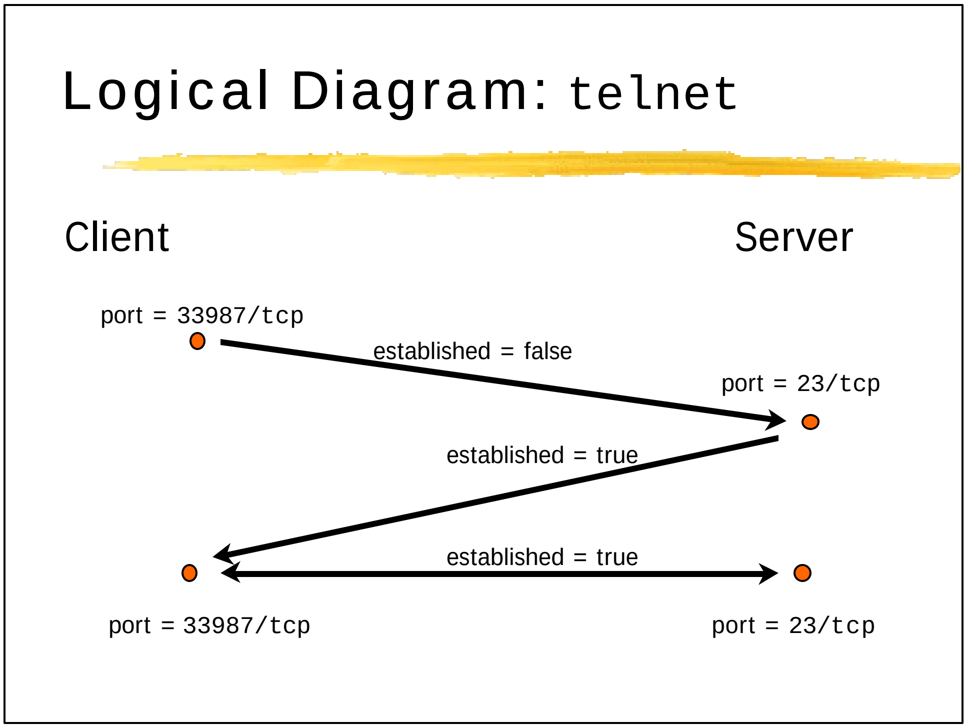
TCP packets have an “established” bit:

- *The first packet in a session has this bit off*
- *The first return packet and all other packets have this bit turned on*
- *Stop new connections by dropping packets that do not have “established” bit on*

TCP connections also have a notion of a session that is “in progress” as opposed to the beginning of a new session. When a TCP client constructs the initial packet of a network connection, it sets a bit in the packet header to “false” indicating that no connection has been established yet. The first packet returned by the remote server has this bit set to “true” indicating that the packet is now part of a session in progress. All other packets sent by client and server also have this bit set to “true”.

Thus, if you want to prevent outsiders from initiating connections, you need to stop packets that have this *established* bit set to false. The problem is that only TCP-based services use this bit. This is why UDP-based services are hard to filter and generally not allowed through firewalls-- there’s no way to tell whether this is a packet that’s part of a session started by a potential hacker on the outside or a legitimate user on the inside.

Note that the technical name for the “established” bit is the *ACK bit*-- the moniker “established bit” derives from the packet filtering syntax for Cisco routers as we will see later.



Logical Diagram: `telnet`

Client

Server

port = `33987/tcp`

established = false

port = `23/tcp`

established = true

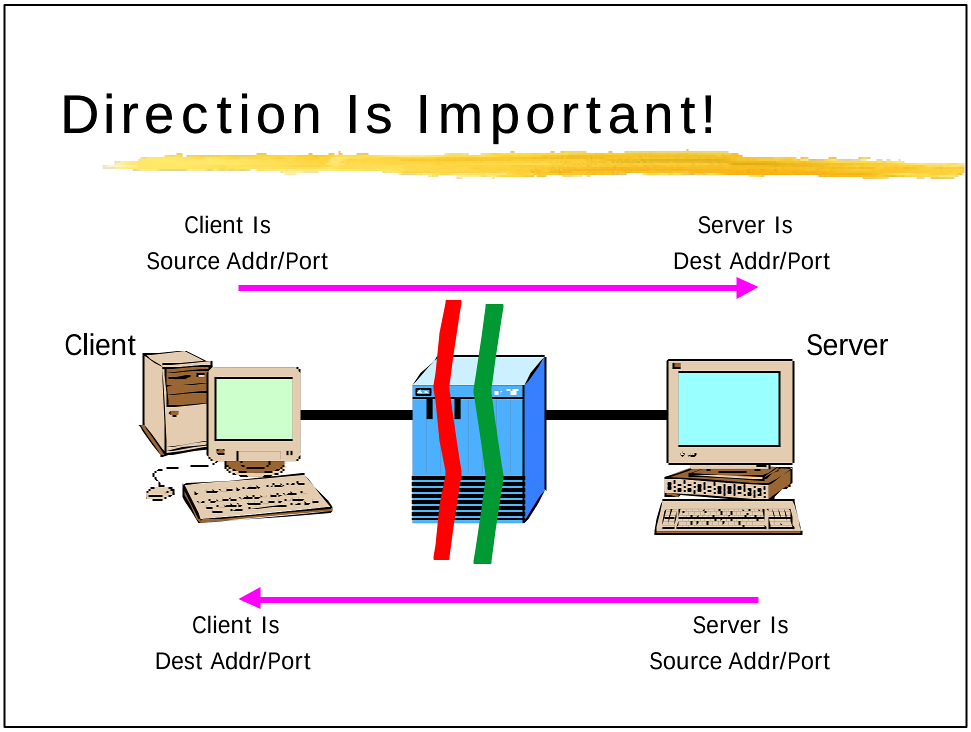
established = true

port = `33987/tcp`

port = `23/tcp`

Here’s our `telnet` example again. Note that the first packet from client to server has the established bit set to false. All subsequent packets have the bit set to true. If our packet filters can stop that first packet, then the client may never initiate a telnet session with the server.

Note that it is possible for an experienced network programmer to generate that same initial packet except that the established bit is set to “true”. However, the server will have no record of an established session and will tear down the connection upon receiving the first packet from the client.



Direction Is Important!

Client Is
Source Addr/Port

Server Is
Dest Addr/Port

Client

Server

Client Is
Dest Addr/Port

Server Is
Source Addr/Port

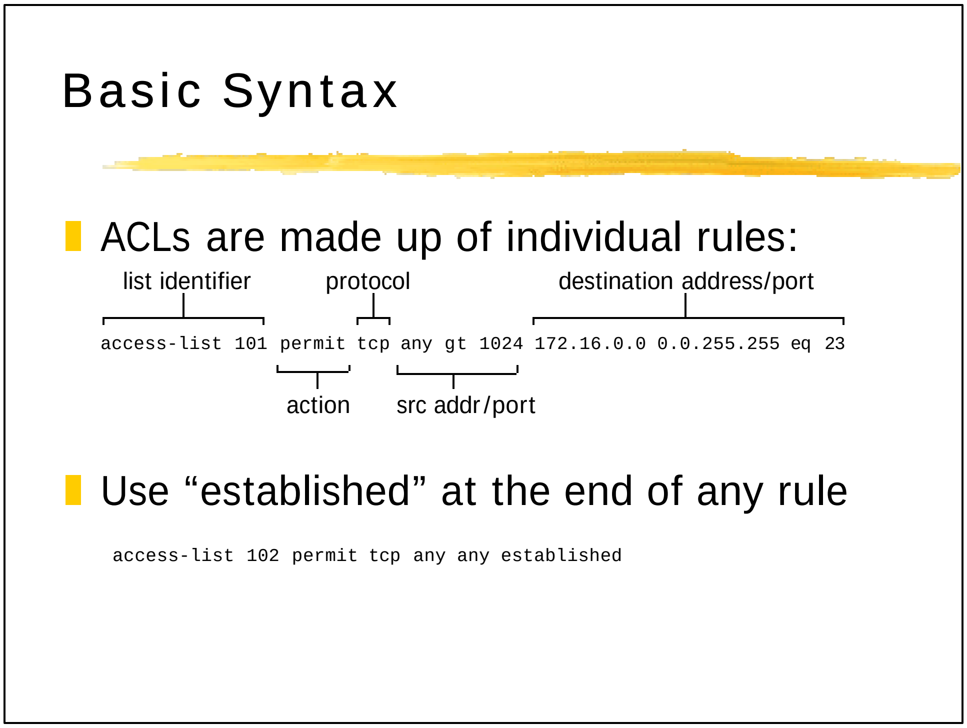
Typically the administrator configures separate filters for packets leaving the organization and for packets coming in from the Internet. It is important to remember that the source and destination address and port information changes depending on which direction the packets are flowing in.

Note that for the rest of the examples in this talk, the internal machine (client or server) or network will be represented to the left of the firewall device and the external machine/network on the right.

Cisco Access Control Lists

Cisco's got the market share in the router business. Best to learn packet filtering on this platform first.

Note that there is substantial difference in packet filtering syntax from one vendor to another, but the core concepts are the same.



Basic Syntax

- ACLs are made up of individual rules:

list identifier protocol destination address/port

```
access-list 101 permit tcp any gt 1024 172.16.0.0 0.0.255.255 eq 23
```

action src addr/port

- Use “established” at the end of any rule

```
access-list 102 permit tcp any any established
```

Typical access lists can contain dozens or even hundreds of individual *access-list* statements. The largest access list your author has seen included over fourteen hundred individual rules.

Each access list begins with an identifying number which groups individual rules into a single access list. Any number `100` or above may be used for Cisco extended access lists (lower numbers are used for basic access lists which have less functionality-- don’t bother).

Next comes the action. `permit` means allow matching packets, `deny` means drop the offending packet and do not allow it to reach its destination.

Protocol can be `tcp`, `udp`, `icmp`, `gre`, etc. The protocol `ip` matches all protocols.

Next comes a source address followed by a port specifier and then a destination address and port specifier. Addresses can look like

`172.16.0.0 0.0.255.255`	*network address*
`host 172.16.1.1`	*individual host*
`any`	*match any addr*

Port specifiers are generally a comparison operator (e.g. `gt` for greater than, `lt` for less than, `eq` for equals) followed by a number. There is also a `range` operator for specifying an inclusive range of ports. The port specifier is always optional.

ACL Pitfalls

- First match and exit behavior
- Every ACL ends with an implicit “deny all”:

```
access-list xxx deny ip any any
```

- You can’t add rules in the middle of an ACL-- must re-create the entire ruleset

It is important to remember that the router stops processing access lists as soon as it finds a rule which matches. *Order is important!* Getting rules out of order can allow traffic that you thought was denied.

Cisco access lists are always in default deny mode. You can stop this behavior by putting

```
access-list xxx permit ip any any
```

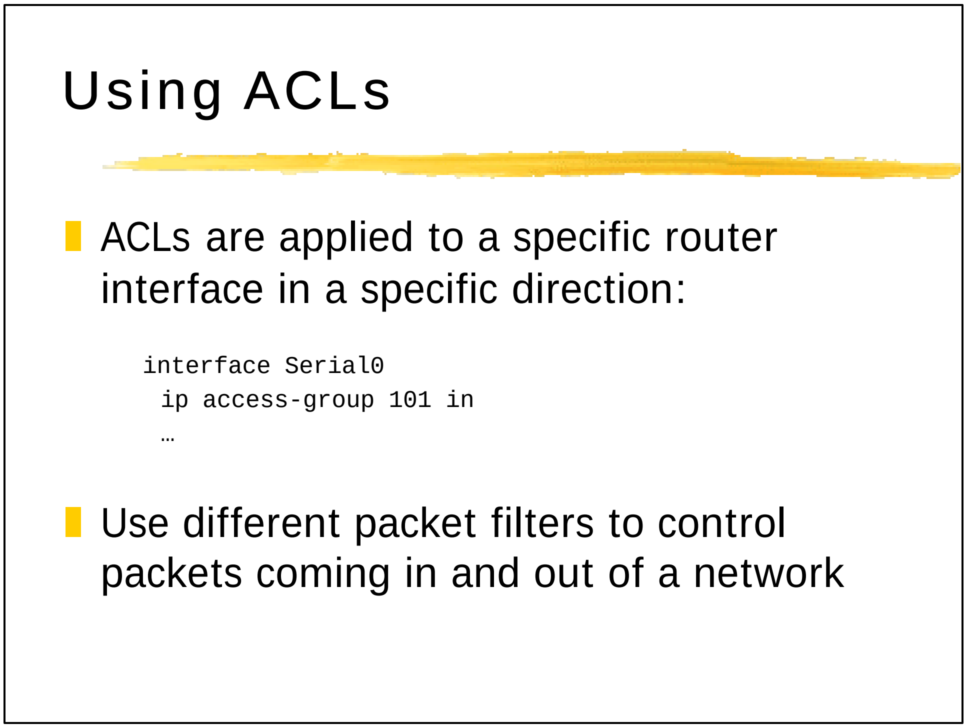
as the last explicit rule in any access list.

Cisco routers do not allow you to edit your access lists and insert rules in the middle. You must destroy and recreate a new access list to insert rules. I recommend putting each access list in a file with the following preamble:

```
no access-list 101
access-list 101 …
access-list 101…
```

You can then use `configure network` to read in this file as an atomic operation and update your ACL.

Over the years, many bugs have been reported against Cisco’s packet filtering functionality-- particularly related to handling of the established bit. Note that this is a good reason to trust Cisco’s packet filtering-- we think we’ve gotten all of th bugs out at this point. Still, make sure you keep up to date on the latest stable IOS release for your platform.



Using ACLs

- ACLs are applied to a specific router interface in a specific direction:

```
interface Serial0
 ip access-group 101 in
 …
```

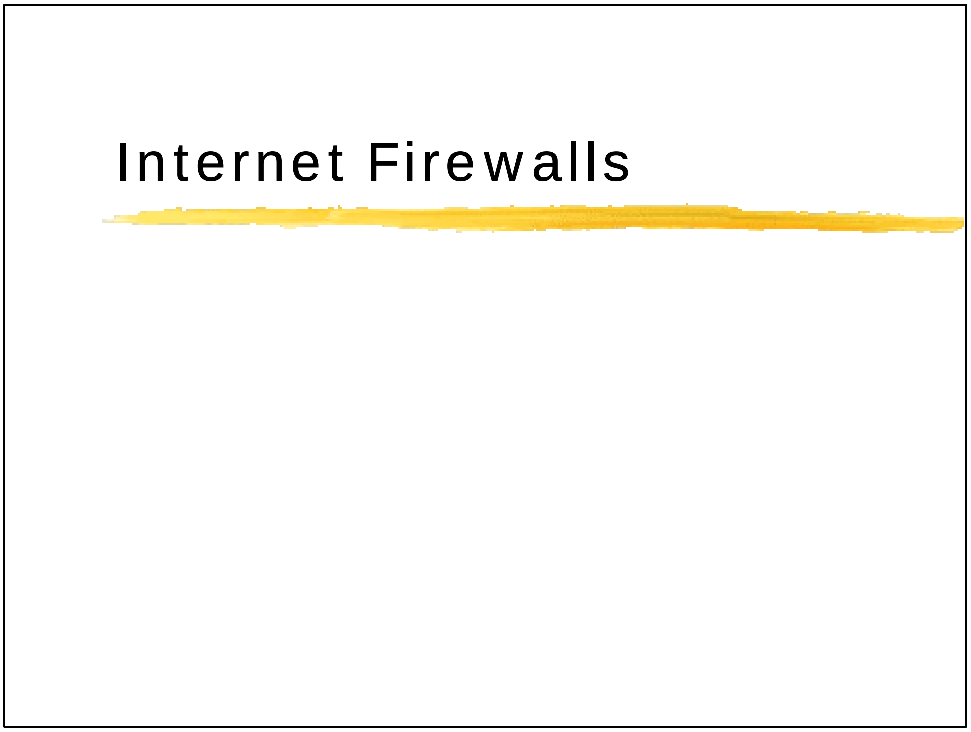
- Use different packet filters to control packets coming in and out of a network

Once you define a complete access list, it must be assigned to a given interface. A given access list may be used by multiple interfaces and a given interface may use multiple access lists.

Note that access lists are applied to an interface in a given direction. If the direction is `in` then the access list is evaluated as packets are picked up off the wire, before making it into the router. If the direction is `out` then the access list is evaluated as the packets leave the router.

Most packet filtering routers use one access list to control packets coming from the “interior” network heading for the outside world and another to stop packets coming in from outside.

Note that inward packet filtering is a relatively new development (circa IOS version 10.0). In most cases, you can substitute an inbound filter on one interface with an outbound filter on another interface.



Internet Firewalls

As an example of how to create access lists, we will be generating a small but usable firewall access-list.

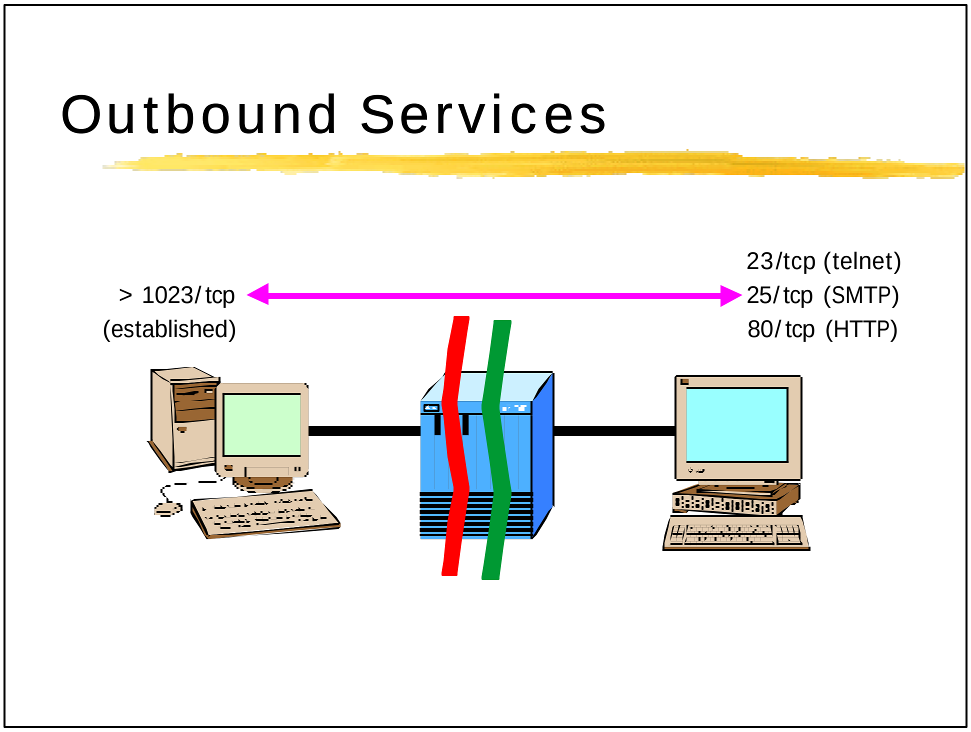
What You Need to Allow

- Outgoing Web, Email, **`telnet`**, etc.
- Incoming Email, Web?, USENET?
- Outgoing FTP (Incoming also?)
- Ping, Traceroute
- DNS

There is a tension when building a firewall between the desire to be secure and the desire of your user community to try out spiffy new network protocols. If your firewall is too draconian then users will go around your firewall with modems and other access.

At a minimum you have to let users get out on the Internet and use common clients like Web browsers, `telnet`/SSH, FTP, etc. There also has to be a way for email and USENET to get in from the outside and to let external customers browse your Web site. It's also useful to let network admins use `ping` and `traceroute` from inside your enterprise, but it's not useful to allow outsiders to map you networks with these tools.

Underlying all Intenet protocols is DNS. DNS is invisible until it stops working, so one of our goals is to see that it remains invisible.



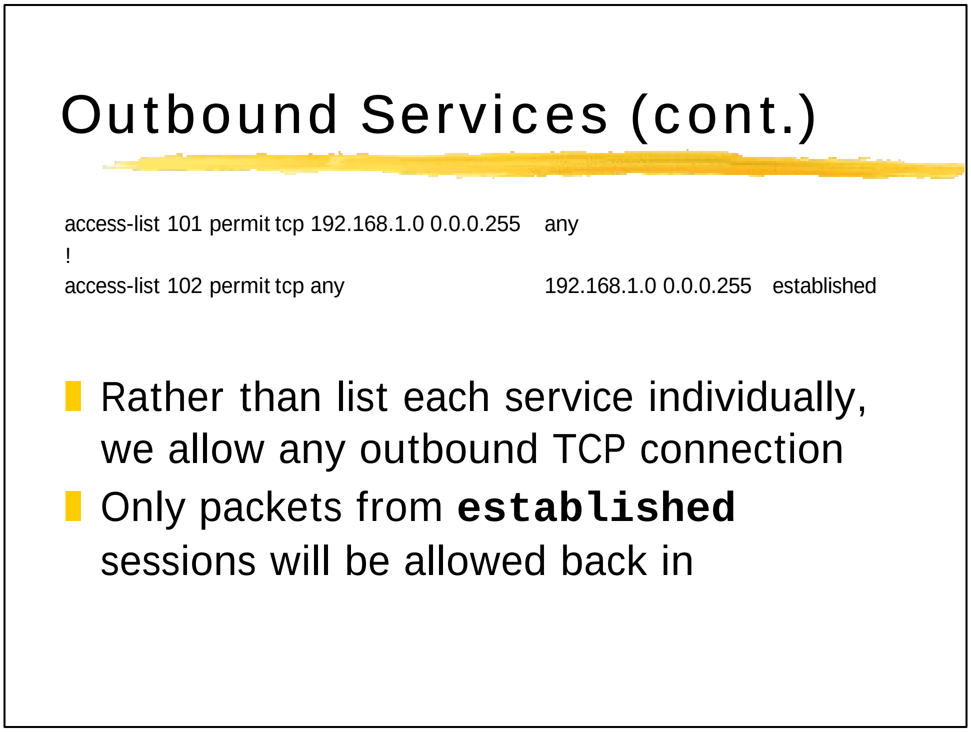
Outbound Services

> 1023/tcp
(established)

23/tcp (telnet)
25/tcp (SMTP)
80/tcp (HTTP)

We saw an example of a telnet session earlier in the talk-- other TCP-based services like SMTP (email) and HTTP (the Web) behave similarly. FTP *does not* behave like `telnet` as we will see later.

For simplicity's sake, the connections are represented here with a single line. The client grabs a high-order TCP port and sends an initial packet to the appropriate service on the remote machine. All packets coming back should have the established bit set.

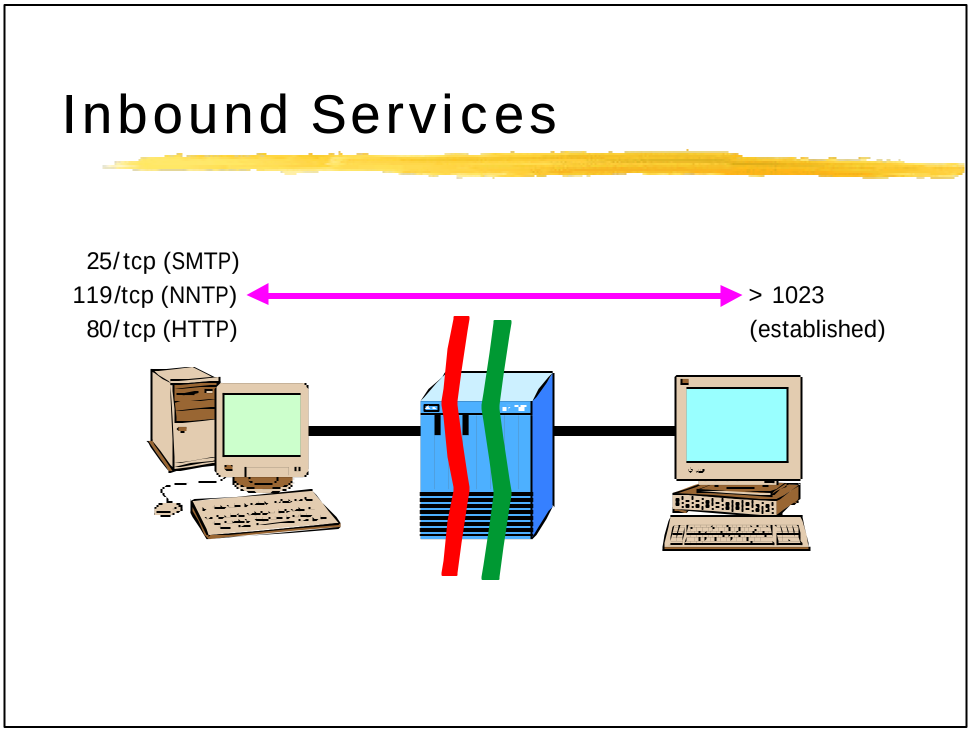


Outbound Services (cont.)

```
access-list 101 permit tcp 192.168.1.0 0.0.0.255    any
!
access-list 102 permit tcp any                      192.168.1.0 0.0.0.255    established
```

- Rather than list each service individually, we allow any outbound TCP connection
- Only packets from **`established`** sessions will be allowed back in

Rather than try to enumerate all of the possible TCP-based services your users could demand, the easiest thing is to simply allow all outgoing TCP sessions from your network(s). This seems like a reasonable security vs. ease of administration trade-off and reduces load on the packet filtering device as well since fewer rules will have to be evaluated.



Inbound Services

25/tcp (SMTP)
119/tcp (NNTP)
80/tcp (HTTP)

> 1023
(established)

To allow connections in from the outside, the sense of the last example is reversed. Connections will be initiated from high-order ports on external machines and come into your network with the established bit *off*.

Inbound Services (cont.)

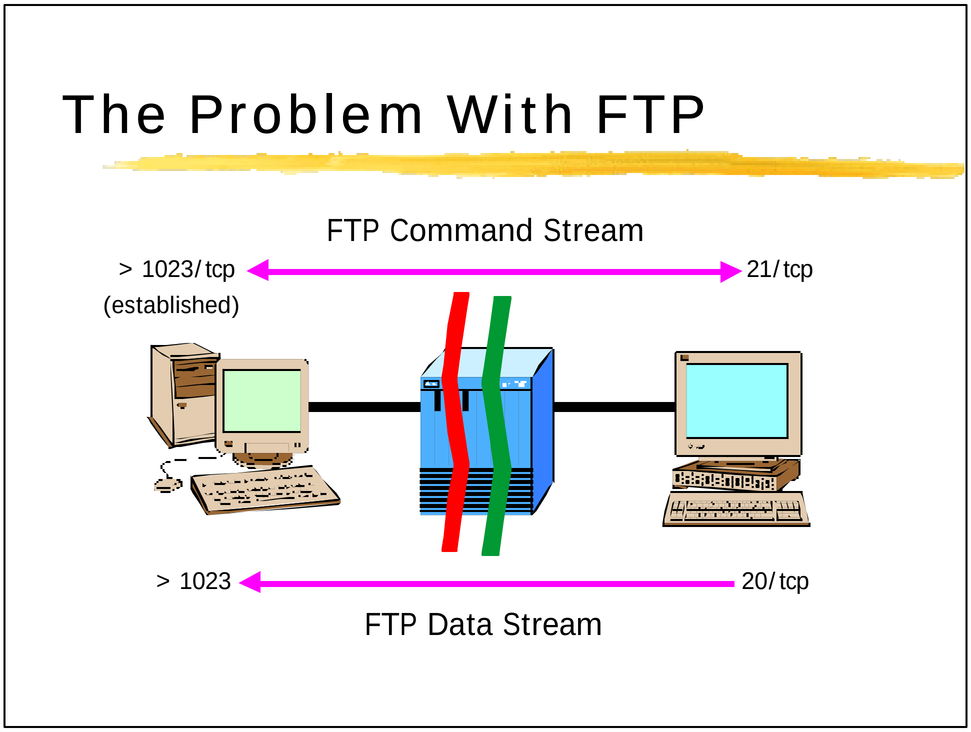
```
access-list 101 permit tcp host 192.168.1.1    any                established
!
access-list 102 permit tcp any                 host 192.168.1.1   eq 25
access-list 102 permit tcp any                 host 192.168.1.1   eq 80
access-list 102 permit tcp host 172.16.1.10    host 192.168.1.1   eq 119
```

- Be as specific as possible when granting access to outsiders
- **`access-list 101`** rule is redundant because of rules for outgoing services

The general rule in any firewall configuration is that outsiders are completely untrustworthy, their systems are a haven for malicious system crackers, and every network between you and them is being watched by packet sniffers (paranoia is a job requirement for Security Admins). External folks should only be granted access to the limited systems and services that they absolutely have to have. Better still is to locate "public" services like your Web and FTP servers on an external network which is on the opposite side of your firewall from your "internal" network.

In the example above, we're saying that the internal host with address `192.168.1.1` is the organization's mail, news, and Web server. Any external machine is allowed to send mail to this machine and browse the Web server, but only our upstream news feed can reach us on the NNTP port. Note that restricting NNTP access to a single host isn't a total win since the source address could be spoofed or the external news server could have been broken into.

Strong authentication and encryption are always a good idea in addition to any packet filtering you may be doing.



The Problem With FTP

FTP Command Stream

> 1023/tcp
(established)

21/tcp

> 1023

20/tcp

FTP Data Stream

The FTP protocol is actually two separate network connections. When you first FTP to a remote server, your client opens the FTP command stream to the remote server on port `21/tcp`. Your login session, and other commands are issued down this connection.

However, every time you want to download or upload a file, or when you want to get a directory listing, the remote server opens a connection to you from port `20/tcp` (the FTP server runs as root), to a random high-order port on your machine that your client has agreed to via the command connection. Your firewall is probably configured to *block* random TCP connections coming in from outside-- which is exactly what the FTP data stream looks like to a stateless firewall.

FTP Problem (cont.)

```
access-list 101 permit tcp 192.168.1.0 0.0.0.255    any
!
access-list 102 permit tcp any eq 20                192.168.1.0 0.0.0.255 gt 1023
```

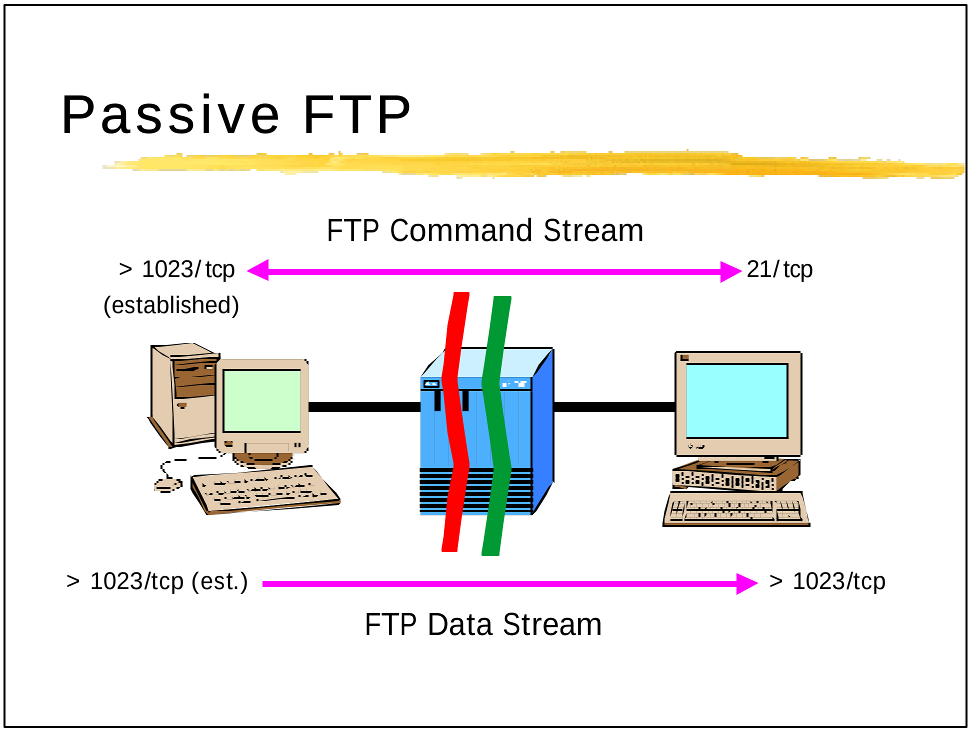
- Source port information could be spoofed: crackers can have access to all high ports!
- Best to use a stateful firewall, an FTP proxy server, or passive FTP

One option is to allow any connection whose source port is `20/tcp`. The problem here is that packet header information can be forged. By creating a packet with the appropriate source port, a malicious outsider can access any high-order TCP port on your network-- X Windows servers, NFS servers, etc.

Stateful firewalls watch the FTP command stream. When you make a download request with your client, the stateful firewall knows to expect a connection from the remote system on the specified port. There is still some possibility that a malicious outsider could jump in and establish their own connection, but they only have access to a single port on your machine and the best they can do is download spurious data to you (which could be very bad if the spurious content is, say, a virus).

FTP proxy servers exist (see `www.socks.nec.com`) which can be located on an external network and safely move FTP traffic through your firewall. Proxy servers typically watch the FTP command stream just like stateful firewalls and know when to expect return data connections.

Passive FTP (see next slide) is also an option...



Passive FTP

FTP Command Stream

> 1023/tcp (established)

21/tcp

> 1023/tcp (est.)

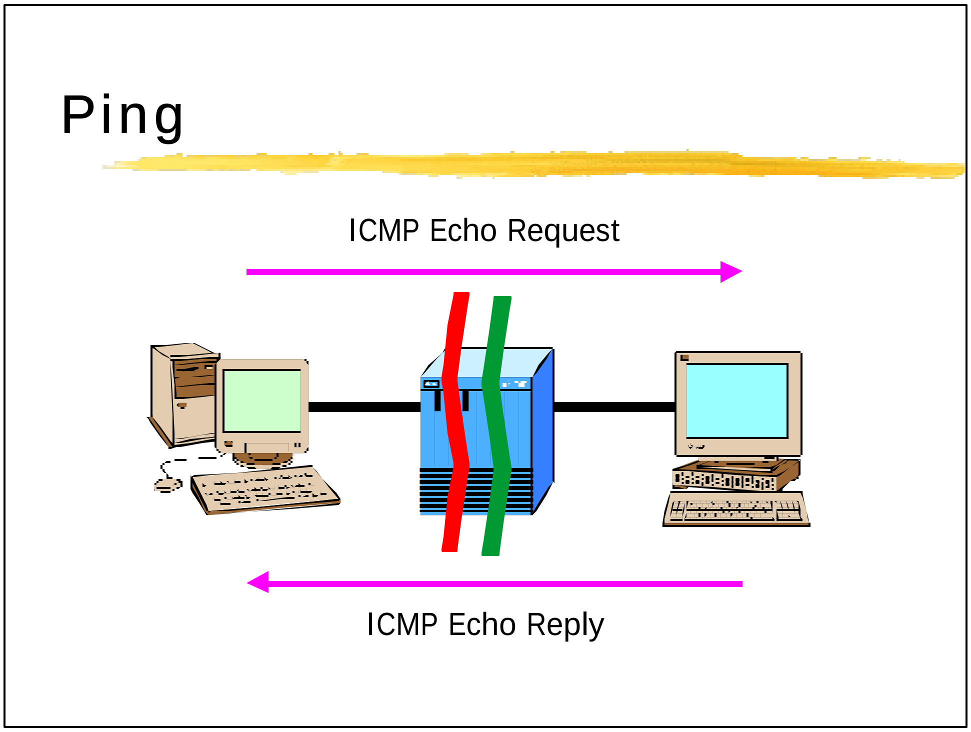
> 1023/tcp

FTP Data Stream

Passive FTP clients work just like standard FTP clients *except* that your client makes the FTP data connection to the remote server, rather than the other way around. This means that the FTP data connection just looks like any other outgoing TCP connection (which we already allowed in an earlier slide).

Web browsers which support FTP are always passive FTP clients. Some other Unix clients (`ncftp`) support PASV mode.

Note that some sites whose FTP servers are behind firewalls do not allow passive FTP connections. They'd have to open all high-order TCP ports to their FTP server through their firewall and some organizations are unwilling to take this risk.



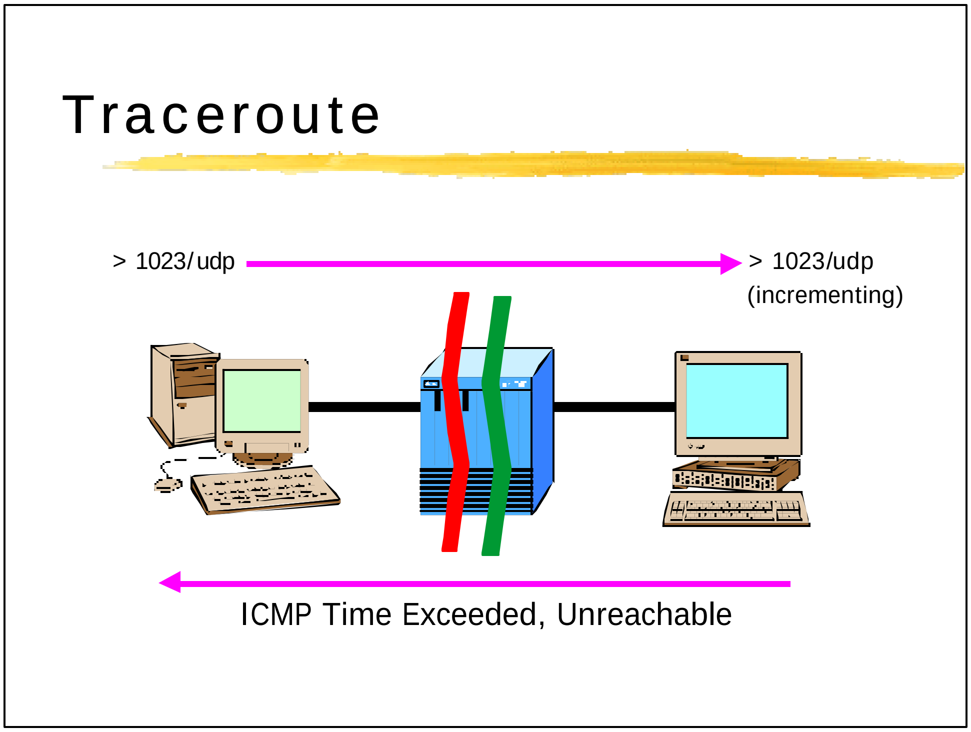
Ping

ICMP Echo Request

ICMP Echo Reply

The `ping` program emits a special ICMP message known as the *echo request* package. At the lowest level of a its network stack, the remote server upon receiving an *echo request* will send back an ICMP *echo reply*.

Cisco packet filters (possibly others) allow you to block ICMP messages by message type. By permitting *echo requests* to go out and *echo replies* to come back in but not vice-versa, you have effectively deployed a one-way `ping` filter. Since there are a number of denial of service type attacks which use ICMP *echo request* packets, blocking incoming instances is a very good thing.



Traceroute

> 1023/udp

> 1023/udp
(incrementing)

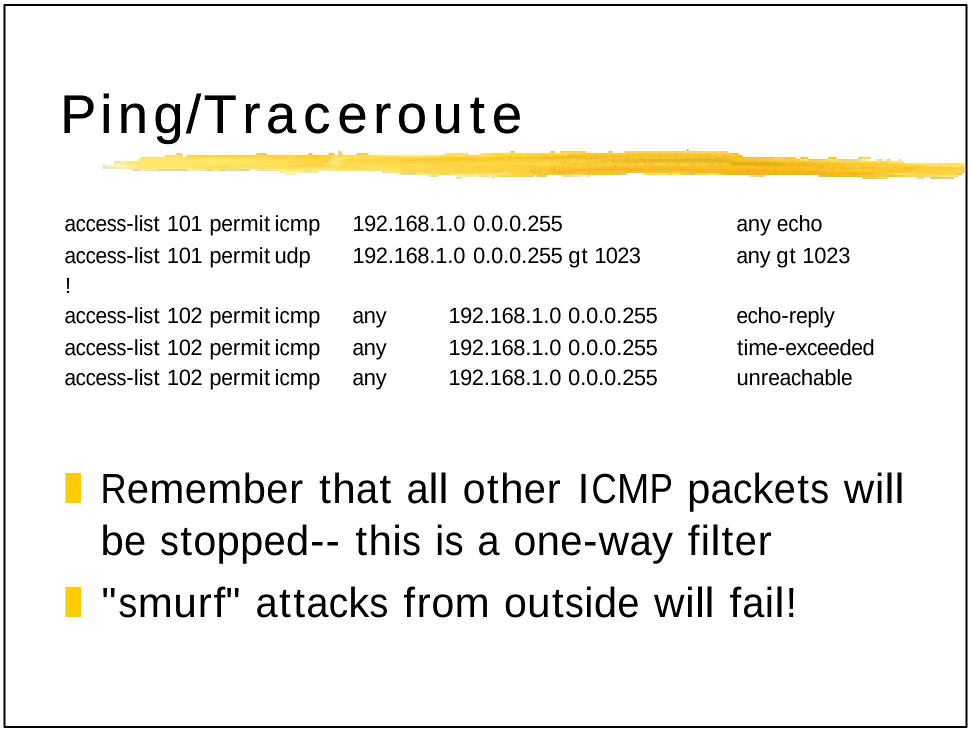
ICMP Time Exceeded, Unreachable

Many administrators think that `traceroute` is also ICMP based. It turns out that `traceroute` is a very cool hack that sends out UDP packets and waits for standard ICMP network errors to come back.

Every IP packet has a "hop count" field. Each time a router forwards a packet, it reduces the hop count in the packet header by one. If this reduces the hop count to zero, the packet is dropped and the router sends back an ICMP *time exceeded* message.

`traceroute` sends out a stream of packets to high-order UDP ports on the remote system. The first packet has its hop count set to one, so the first router that's encountered drops the packet and sends back *time exceeded.* `traceroute` gets the IP address of this router from the header of the ICMP packet and displays it (or the hostname from DNS). The next packet has hop count two, so it gets one router farther before failing. The next packet has hop count three, and so on.

Ultimately, the hop count will be large enough that the packet gets to its destination. At this point, there is usually no server listening on the random port chosen by `traceroute` and the remote machine sends back ICMP *unreachable*. `traceroute` displays the hostname and stops.



Ping/Traceroute

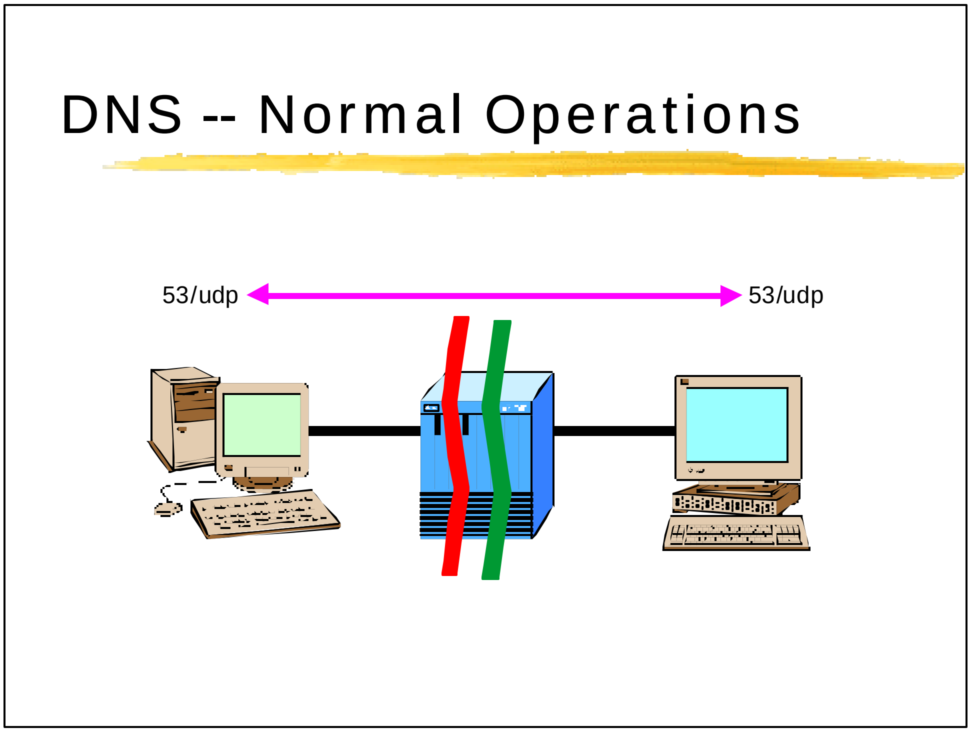
```
access-list 101 permit icmp   192.168.1.0 0.0.0.255                   any echo
access-list 101 permit udp    192.168.1.0 0.0.0.255 gt 1023           any gt 1023
!
access-list 102 permit icmp   any      192.168.1.0 0.0.0.255          echo-reply
access-list 102 permit icmp   any      192.168.1.0 0.0.0.255          time-exceeded
access-list 102 permit icmp   any      192.168.1.0 0.0.0.255          unreachable
```

- Remember that all other ICMP packets will be stopped-- this is a one-way filter
- "smurf" attacks from outside will fail!

For ping, we allow ICMP *echo request* packets to escape from our network and *echo replies* to come back.

For traceroute, we have to allow the outgoing UDP packets to escape and let the *time exceeded* and *unreachable* messages to come back in.

Note that the default deny stance enforced by Cisco routers prevents the same packets from coming in from outside as well as the responses from escaping your network. Other dangerous ICMP packets are also blocked.



DNS -- Normal Operations

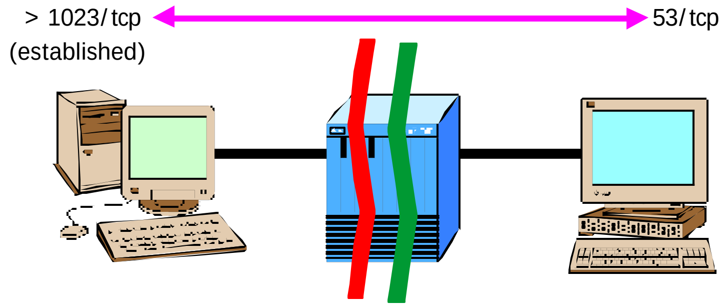
When your local machine needs to look up information via DNS, it typically does a lookup to a local nameserver which is situated behind your firewall on your internal network. If this server doesn't have the information you requested in its cache, it makes "server-to-server" queries to remote nameservers outside your firewall. These queries happen on `53/udp` (both source and destination ports). You should only allow UDP connections on `53/udp` to enable DNS.

Actually, this was the behavior in BIND v4. BIND v8 has changed the default behavior for server-to-server queries: now you local nameserver picks a random port above 1023 for its connections. This breaks most existing firewalls. The

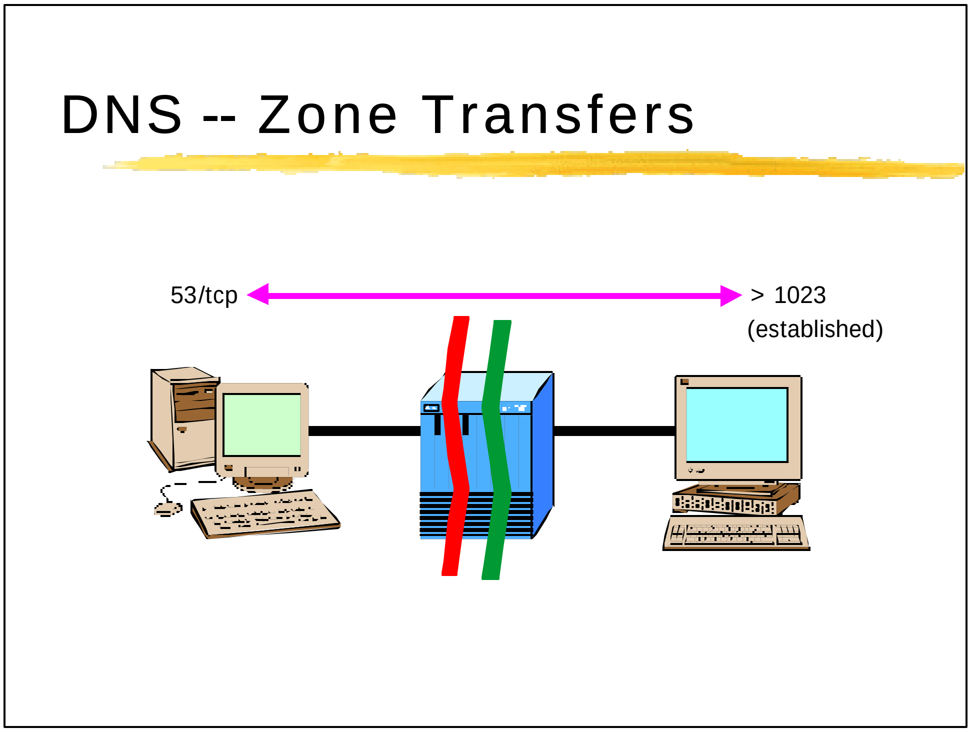
```
query-source address * port 53;
```

line in the `options` block of `named.conf` forces the old `53/udp` for source and destination behavior.

DNS -- Large Response

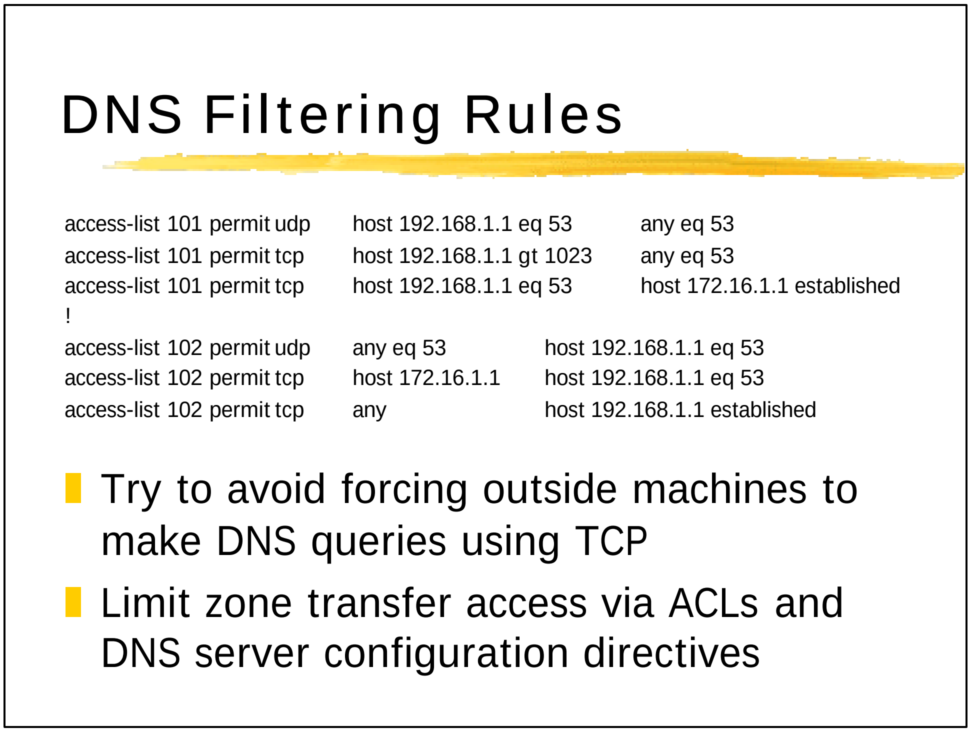


If the remote name server detects that its response to your server-to-server query would be larger than 512 bytes, it sends back a UDP packet which instructs your local nameserver to redo the query using TCP. This is a standard outbound TCP connection: your nameserver grabs a random high-order TCP port and connects to `53/tcp` on the remote system, so TCP retries are already permitted by our previous examples.



DNS -- Zone Transfers

Finally, if you have external secondary nameservers outside your firewall, you have to allow them to transfer your DNS zone files from your primary name server (actually you can have them do zone transfers from a secondary server--possibly located on your external network outside of your firewall?). This is an inbound TCP connection to `53/tcp` on your internal nameserver.



DNS Filtering Rules

```
access-list 101 permit udp   host 192.168.1.1 eq 53     any eq 53
access-list 101 permit tcp   host 192.168.1.1 gt 1023   any eq 53
access-list 101 permit tcp   host 192.168.1.1 eq 53     host 172.16.1.1 established
!
access-list 102 permit udp   any eq 53         host 192.168.1.1 eq 53
access-list 102 permit tcp   host 172.16.1.1   host 192.168.1.1 eq 53
access-list 102 permit tcp   any               host 192.168.1.1 established
```

- Try to avoid forcing outside machines to make DNS queries using TCP
- Limit zone transfer access via ACLs and DNS server configuration directives

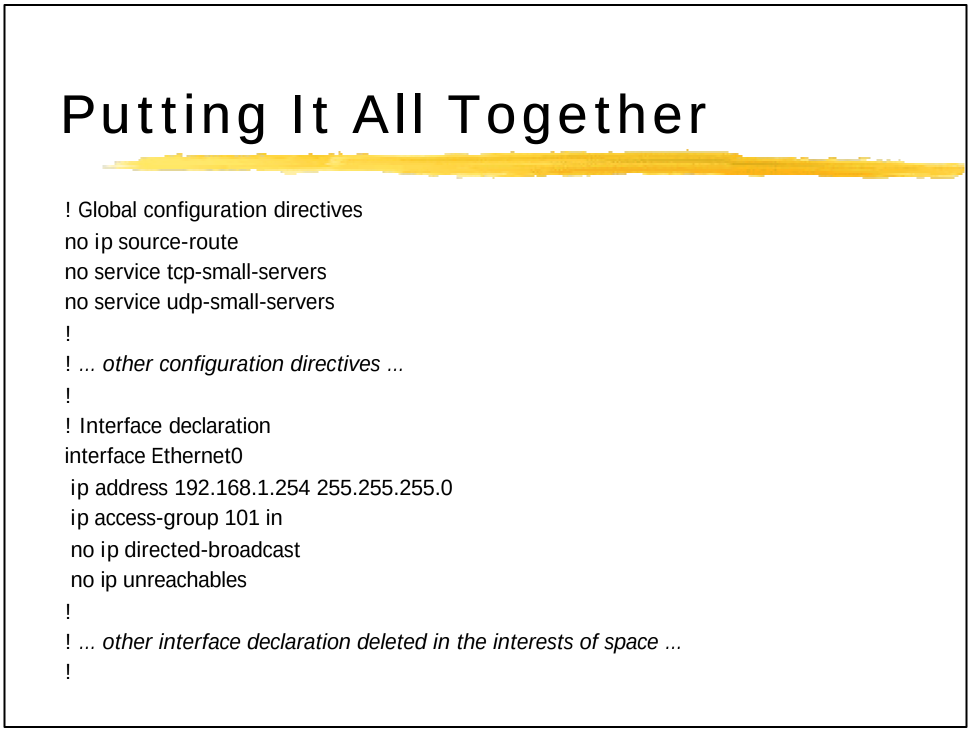
In this example, the host `192.168.1.1` is our internal nameserver and `172.16.1.1` is an external secondary server.

We allow UDP connections on `53/udp` for server-to-server queries between our internal nameserver and any external host. We also allow outgoing TCP connections, though this rule is redundant given our earlier examples. Finally we allow the response packets to zone transfer attempts by our external secondary to escape.

Inbound we need to allow the return UDP packets from our server-to-server queries. We also need to permit the initial zone transfer connection from our remote secondary as well as allow packets from established sessions.

It's probably a good idea to try and prevent outsiders from downloading your entire zone file at one time since it potentially contains a lot of damaging information. This is why we were so specific about access to `53/tcp` in our example. However, if your internal DNS server is going to generate large responses to external queries (e.g. many nameservers for your domain, or a single host with lots of interfaces), you will need to allow everybody in the world to get at `53/tcp` or the TCP-based retries will fail.

While most nameserver can and should be configured to limit zone transfers to known hosts, it's also a good idea to arrange things so that you don't force external nameservers to retry with TCP.



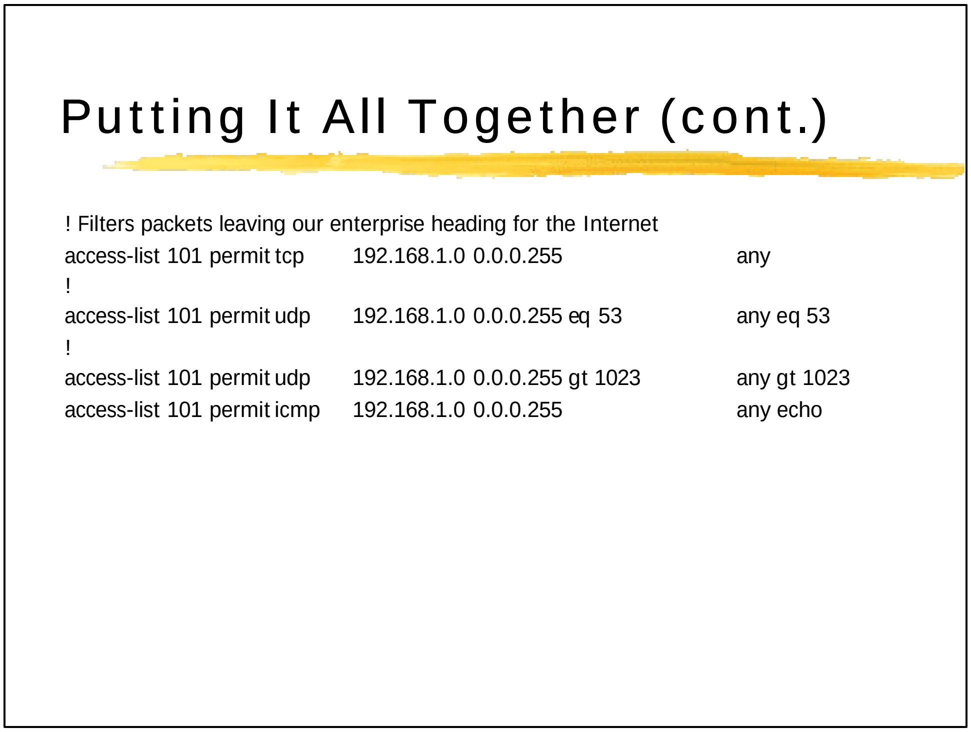
Putting It All Together

```
! Global configuration directives
no ip source-route
no service tcp-small-servers
no service udp-small-servers
!
! ... other configuration directives ...
!
! Interface declaration
interface Ethernet0
 ip address 192.168.1.254 255.255.255.0
 ip access-group 101 in
 no ip directed-broadcast
 no ip unreachables
!
! ... other interface declaration deleted in the interests of space ...
!
```

Before we get to the actual ACLs, it's useful to look at some other configuration directive you can use to improve security on your Cisco.

`no ip source-route` causes your router to drop any packets that come in with source routing information-- typically used these days by people trying to bypass your network security policy. `no service *-small-servers` stops your router from responding to service requests like `echo` and `chargen` which can be used as a denial of service attack on your router.

Under the interface declaration, `no ip directed-broadcast` will cause your router to drop packets headed for the broadcast address on the LAN that router interface is connected to. This can prevent your networks from being used as an intermediate network in a smurf attack. `no ip unreachables` stops your router from sending back ICMP *administratively unreachable* messages when it drops a packet and makes the router just drop packets silently-- useful for not giving away too much detail regarding your firewalling policy.



Putting It All Together (cont.)

```
! Filters packets leaving our enterprise heading for the Internet
access-list 101 permit tcp    192.168.1.0 0.0.0.255              any
!
access-list 101 permit udp    192.168.1.0 0.0.0.255 eq 53        any eq 53
!
access-list 101 permit udp    192.168.1.0 0.0.0.255 gt 1023      any gt 1023
access-list 101 permit icmp   192.168.1.0 0.0.0.255              any echo
```

Merging all of the examples from previous slides in this section and removing redundant rules, we're left with a very simple filter for controlling packets leaving our network for the Internet.

We allow any TCP connection outbound, including `telnet`, HTTP, mail, and DNS queries. We also allow UDP-based server-to-server DNS queries. We also allow the initiating packets for `traceroute` and `ping`.

In general, outbound filters tend to be shorter than inbound filters since we're more lenient with what our internal users can do to the outside world.

Putting It All Together (cont.)

```
! Filters packets coming in from the Internet
access-list 102 deny    ip  192.168.1.0 0.0.0.255              any
access-list 102 deny    ip  127.0.0.0 0.255.255.255            any
!
access-list 102 permit tcp any                      192.168.1.0 0.0.0.255   established
access-list 102 permit tcp any                      host 192.168.1.1        eq 25
access-list 102 permit tcp host 172.16.1.10         host 192.168.1.1        eq 119
!
access-list 102 permit udp      any eq 53           host 192.168.1.1    eq 53
access-list 102 permit tcp      host 172.16.1.1     host 192.168.1.1    eq 53
!
access-list 102 permit icmp     any       192.168.1.0 0.0.0.255         echo-reply
access-list 102 permit icmp     any       192.168.1.0 0.0.0.255         time-exceeded
access-list 102 permit icmp     any       192.168.1.0 0.0.0.255         unreachable
```

[Interface declaration deleted in the interests of space]

The first rules in the incoming packet filter are new. These are generally referred to as "anti-spoofing" rules and should be at the beginning of any inbound access list. They prevent outsiders from injecting spoofed packets that pretend to be from hosts on your internal network. They can also warn you if there's a back-door to your network that's leaking packets out onto the Internet.

Next we have the rule which permits packets from established TCP sessions to come back into hosts on your internal network. This rule will hit about 95% of the packets coming into your network so it's best for performance (remember: "first match and exit") to put it as high up in the list as possible (right after the anti-spoofing filters. This is the only way you can have 1400 lines of access list and prevent your router from choking.

Next we allow inbound SMTP connections from anywhere, and NNTP connections (USENET) from our upstream newsfeed. Next we permit DNS server-to-server queries and zone transfers (only from our external secondary).

Finally we allow the responses from our `ping`s and `traceroute`s.

Final Thoughts

Some final notes that weren't able to be worked in elsewhere or which need special emphasis.

Caveats

- Packet filters are a good first-line defense
- Need to be combined with strong authentication and encryption solutions
- Cannot help you protect poorly designed network protocols (e.g. H.323 services)
- Won't help protect you from active content (virii, Java, ActiveX, etc.)

While using packet filters can significantly improve the security of your organization, they are not a complete solution. Strong authentication and solid encryption are needed-- particularly if you allow your users to get into internal machines from outside your firewall. Look into SSH, one-time passwords, and other security tools. Note that many routers include VPN functionality which can be used to safely route traffic over public (hostile) networks.

Some network protocols simply cannot be handled by simple packet filters. Sometimes this is simply bad design. In any event, you will need a stateful firewall or a proxy server to safely allow these services for internal use. Some network scans and other attacks can bypass the packet filters we've implemented in this class. Bottom line is that you probably will need a modern, stateful firewall for at least your Internet connection.

On the other hand, stateful firewalls can be a significant performance bottleneck. Consider delpoying packet filters both in front of and behind your stateful firewall. Every packet stopped by the packet filter is one less packet that has to get considered by your stateful device and overall things will run much faster. Unfortunately you lose out on the stateful firewall device being the sole point of logging for network events.

You may also want to employ solutions for filtering active content from Web pages and email messages.

Internal Firewalls

- Packet filters can also be used to protect interior networks from each other
- Good for stopping "packets o' death" from escaping testing labs
- Probably should be in "default permit" stance, rather than "default deny"

While this course concentrated on using packet filters as an Internet firewall, simple packet filters can and should be used inside your company to prevent obviously broken packets from causing problems on your network. This is particularly important if part of your company develops networking products.

In general these filters are set up to either allow a very large generic group of hosts (e.g., all valid addresses in your organization), or are in a "default permit" stance where every packet is allowed except certain known "bad packets". It may be hard to enumerate all of the possible "bad" packets that could be generated-- "default deny" may be safer.

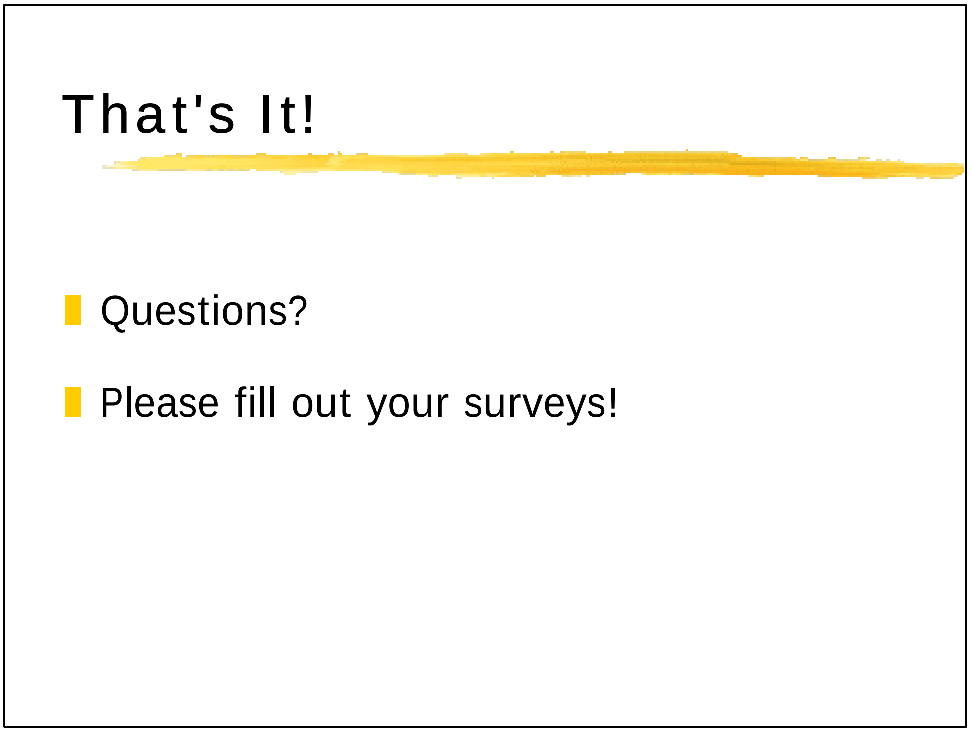
Stateful Firewalls

- Stateful firewalls inspect packet contents and understand application stream
- Permit return FTP data stream or other external connects based on previous state
- Have to be extended on a protocol by protocol basis-- some development lag
- Often run on Unix/NT platforms, so platform security becomes an issue

In general, stateful packet filters perform some level of content analysis on packets as they pass through the firewall. For example, the firewall will reconstruct the FTP command stream and look for attempts to download files or get directory listings. With this information, the firewall can prepare to accept the return FTP data connection from the remote host.

The difficulty is that each protocol has to be coded into the firewall's state engine separately. This means a 3-6 month minimum lag time before the firewall is ready to deal intelligently with a new protocol (though basic packet filtering type functionality will be available, even in stateful firewalls).

Note also that these firewall products are relatively new, fairly complex pieces of software running on multi-user computing platforms (unlike, say, a router). Even assuming the firewall software itself is bug-free, security shortcomings in the underlying OS platform can allow crackers to subvert your firewall and break into your enterprise.



That's It!

- Questions?
- Please fill out your surveys!

This space intentionally left blank.