Game Plan

• Network Attacks:
  – **DHCP**: protocol for *bootstrapping* Internet access

• Firewalls: Controlling networks
  – (on the cheap!)

• Users/applications *subverting* (sneaking around) firewalls
  – (as time permits)
Internet Bootstrapping: DHCP

- New host doesn’t have an IP address yet
  - So, host doesn’t know what source address to use

- Host doesn’t know who to ask for an IP address
  - So, host doesn’t know what destination address to use

- Solution: shout to “discover” server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address

DHCP = Dynamic Host Configuration Protocol
Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)

DHCP server

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)

DNS server = system used by client to map hostnames like gmail.com to IP addresses like 74.125.224.149

Gateway router = router that client uses as the first hop for all of its Internet traffic to remote hosts
Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)
Dynamic Host Configuration Protocol

The process involves a new client requesting configuration from a DHCP server:

- **DHCP discover** (broadcast)
- **DHCP offer**
- **DHCP request** (broadcast)
- **DHCP ACK**

The "offer" message includes an IP address, DNS server, "gateway router", and how long the client can have these ("lease" time).

**Threats?**
Dynamic Host Configuration Protocol

- DHCP discover (broadcast)
- DHCP offer
- DHCP request (broadcast)
- DHCP ACK

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time).

Local attacker on same subnet can hear new host’s DHCP request.

New client

DHCP server
Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)

DHCP server

DHCP offer

DHCP request (broadcast)

DHCP ACK

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)

Attacker can race the actual server; if attacker wins, replaces DNS server and/or gateway router
DHCP Threats

• Substitute a fake DNS server
  – Redirect any of a host’s lookups to a machine of attacker’s choice (e.g., gmail.com = 6.6.6.6)

• Substitute a fake gateway router
  – Intercept all of a host’s off-subnet traffic
    o (even if not preceded by a DNS lookup)
  – Relay contents back and forth between host and remote server
    o Modify however attacker chooses
  – This is one type of invisible *Man In The Middle* (MITM)
    o Victim host generally has no way of knowing it’s happening! :-(
    o (Can’t necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)

• How can we fix this? **Hard**
Summary: DHCP Security Issues

• DHCP threats highlight:
  – Broadcast protocols inherently at risk of local attacker spoofing
    o Attacker knows exactly when to try it …
    o … and can see the victim’s messages
  – When initializing, systems are particularly vulnerable because they can lack a trusted foundation to build upon
  – Tension between wiring in trust vs. flexibility and convenience
  – MITM attacks insidious because no indicators they’re occurring
Motivation: How do you harden a set of systems against external attack?
- **Key Observation:**
  - *The more network services your machines run, the greater the risk*
  - Due to larger attack surface

One approach: on each system, turn off unnecessary network services
- But you have to know *all* the services that are running
- And sometimes some trusted remote users still require access

Plus key question of **scaling**
- What happens when you have to secure 100s/1000s of systems?
- Which may have different OSs, hardware & users …
- Which may in fact not all even be identified …
Taming Management Complexity

- Possibly more scalable defense: Reduce risk by blocking *in the network* outsiders from having unwanted access your network services
  - Interpose a **firewall** the traffic to/from the outside must traverse
  - **Chokepoint** can cover 1000s of hosts
    - Where in everyday experience do we see such chokepoints?
Selecting a Security Policy

- Effectiveness of firewall relies on deciding what policy it should implement:
  - Who is allowed to talk to whom, accessing what service?

- Distinguish between inbound & outbound connections
  - Inbound: attempts by external users to connect to services on internal machines
  - Outbound: internal users to external services
  - Why? Because fits with a common threat model

- Conceptually simple access control policy:
  - Permit inside users to connect to any service
  - External users restricted:
    - Permit connections to services meant to be externally visible
    - Deny connections to services not meant for external access
How To Treat Traffic Not Mentioned in Policy?

• **Default Allow**: start off permitting external access to services
  – Shut them off as problems recognized
How To Treat Traffic Not Mentioned in Policy?

- **Default Allow**: start off permitting external access to services
  - Shut them off as problems recognized

- **Default Deny**: start off permitting just a few known, well-secured services
  - Add more when users complain (and mgt. approves)
How To Treat Traffic Not Mentioned in Policy?

• **Default Allow:** start off permitting external access to services
  – Shut them off as problems recognized

• **Default Deny:** rt off permitting just a few known, well-secured services
  – Add more when users complain (and mgt. approves)

• Pros & Cons?
  – Flexibility vs. conservative design
  – Flaws in Default Deny get noticed more quickly / less painfully

**In general, use Default Deny**
Packet Filters

• Most basic kind of firewall is a packet filter
  – Router with list of access control rules
  – Router checks each received packet against security rules to decide to forward or drop it
  – Each rule specifies which packets it applies to based on a packet’s header fields (stateless)
    • Specify source and destination IP addresses, port numbers, and protocol names, or wild cards
Packet Filters

- Most basic kind of firewall is a *packet filter*
  - Router with list of access control rules
  - Router checks each received packet against security rules to decide to forward or drop it
  - Each rule specifies which packets it applies to based on a packet’s header fields (*stateless*)
    - Specify source and destination IP addresses, port numbers, and protocol names, or wild cards
    - Each rule specifies the *action* for matching packets: **ALLOW** or **DROP** (aka DENY)
      
      `<ACTION> <PROTO> <SRC:PORT> -> <DST:PORT>`
  
  - First listed rule has *precedence*
Examples of Packet Filter Rules

allow tcp 4.5.5.4:1025 -> 3.1.1.2:80

• States that the firewall should **permit** any TCP packet that’s:
  – from Internet address 4.5.5.4 **and**
  – using a source port of 1025 **and**
  – destined to port 80 of Internet address 3.1.1.2

deny tcp 4.5.5.4:* -> 3.1.1.2:80

• States that the firewall should **drop** any TCP packet like the above, regardless of source port
Examples of Packet Filter Rules

```
deny tcp 4.5.5.4:* -> 3.1.1.2:80
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
```

- *In this order*, the rules won’t allow *any* TCP packets from 4.5.5.4 to port 80 of 3.1.1.2

```
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
deny tcp 4.5.5.4:* -> 3.1.1.2:80
```

- *In this order*, the rules allow TCP packets from 4.5.5.4 to port 80 of 3.1.1.2 *only* if they come from source port 1025
5 Minute Break

Questions Before We Proceed?
Expressing Policy with Rulesets

• Goal: prevent *external access* to Windows SMB (TCP port 445)
  – Except for one special external host, 8.4.4.1

• Ruleset:

  ```
  allow tcp 8.4.4.1:* -> *:445
  drop tcp *:* -> *:445
  allow * *:* -> *:*
  ```

• Problems?
  – No notion of *inbound* vs *outbound* connections
    • Drops outbound SMB connections from inside users
  – (This is a *default-allow* policy!)
Expressing Policy with Rulesets, con’t

• Want to allow:
  – Inbound mail connections to our mail server (1.2.3.4:25)
  – All outbound connections from our network, 1.2.3.0/24
    • 1.2.3/24 = “any address for which the top 24 bits match 1.2.3.0”
    • So it ranges from 1.2.3.0, 1.2.3.1, ..., 1.2.3.255
  – Nothing else

• Consider this ruleset:

```plaintext
allow tcp *:* -> 1.2.3.4:25
allow tcp 1.2.3.0/24:* -> *:*
drop  *   *:* -> *:*
```

• This policy doesn't work ...
  – TCP connections are bidirectional
  – 3-way handshake: client sends SYN, receives SYN+ACK, sends ACK
    • Followed by either/both sides sending DATA (w/ ACK bit set)
Problem: Outbound Connections Fail

1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:* 
3. drop * *:* -> *:* 

• Inside host opens TCP connection to port 80 on external machine: 
  – Initial SYN packet passed through by rule 2 
  – SYN+ACK packet coming back is dropped 
    • Fails rule 1 (not destined for port 25) 
    • Fails rule 2 (source not inside host) 
    • Matches rule 3 ⇒ DROP
Problem: Outbound Connections Fail

1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:*  
3. drop * *:* -> *:*  

• Fix?
  – In general, we need to distinguish between 2 kinds of inbound packets
    • Allow inbound packets associated with an outbound connection
    • Restrict inbound packets associated with an inbound connection
  – How do we tell them apart?
    • Approach #1: remember previous outbound connections
      – Requires state :-(
    • Approach #2: leverage details of how TCP works ...
Inbound vs. Outbound Connections

• Key TCP feature: ACK bit set on all packets except first
  – **Plus**: TCP receiver disregards packets with ACK set if they don’t belong to an existing connection

• Solution ruleset:

  1. `allow tcp *:* -> 1.2.3.4:25`
  2. `allow tcp 1.2.3.0/24:* -> *:*`
  3. `allow tcp *:* -> 1.2.3.0/24:* only if ACK bit set`
  4. `drop * *:* -> *:*`

  – Rules 1 and 2 allow traffic in either direction for **inbound** connections to port 25 on machine **1.2.3.4**
  – Rules 2 and 3 allow **outbound** connections to any port
How This Ruleset Protects

1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:*  
3. allow tcp *:* -> 1.2.3.0/24:* only if ACK bit set
4. drop * *:* -> *:*  

• Suppose external attacker tries to exploit vulnerability in SMB (TCP port 445):
  = Attempts to open an inbound TCP connection to internal SMB server
• Attempt #1: Sends SYN packet to server
  – Packet lacks ACK bit ⇒ no match to Rules 1-3, dropped by Rule 4
• Attempt #2: Sends SYN+ACK packet to server
  – Firewall permits the packet due to Rule 3
  – But then dropped by server’s TCP stack (since ACK bit set, but isn’t part of existing connection)
Subverting Firewalls

• Along with possible bugs, packet filters have a fundamentally limited semantic model
  – They lack a full understanding of the meaning of the traffic they carry
    o In part because operate only at layers 3 & 4; not 7

• How can a local user who wants to get around their site’s firewall exploit this?
  – (Note: we’re not talking about how an external attacker can escape a firewall’s restrictions)

• One method of subversion: abuse ports
  – Who says that e.g. port 22/tcp = SSH?
    o Why couldn’t it be say Skype or BitTorrent?
    o Just requires that client & server agree on app proto
Hiding on Other Ports

• Method #1: use port allocated to another service (how can this be detected?)

• Method #2: tunneling
  – Encapsulate one protocol inside another
  – Receiver of “outer” protocol decapsulates interior tunneled protocol to recover it
  – Pretty much any protocol can be tunneled over another (with enough effort)

• E.g., tunneling IP over SMTP
  – Just need a way to code an IP datagram as an email message (either mail body or just headers)
Example: Tunneling IP over Email

From: doesn't-matter@bogus.com
To: my-buddy@tunnel-decapsulators.R.us
Subject: Here’s my IP datagram

IP-header-version: 4
IP-header-len: 5
IP-ID: 11234
IP-src: 1.2.3.4
IP-dst: 5.6.7.8
IP-payload: 0xa144bf2c0102...

Remote email server receives this legal email, builds an IP packet corresponding to description in email body … … and injects it into the network

How can a firewall detect this??

This operator of this email server has chosen to cooperate with the email sender to help them tunnel
Tunneling, con’t

• E.g., IP-over-ICMP:
  – Embed IP datagram as the payload of a “ping” packet

• E.g., Skype-over-HTTP:
  – Encode Skype messages in URL of requests and header fields of replies

• Note #1: to tunnel, the sender and receiver must both cooperate (so it’s not useful for initial attacks)

• Note #2: tunneling has many legitimate uses too
  – E.g., Virtual Private Networks (VPNs)
    o Make a remote machine look like it’s local to its home network
    o Tunnel encrypts traffic for privacy & to prevent meddling
Secure External Access to Inside Machines

- Often need to provide secure remote access to a network protected by a firewall
  - Remote access, telecommuting, branch offices, …
- Create secure channel (*Virtual Private Network*, or **VPN**) to tunnel traffic from outside host/network to inside network
  - Provides **Authentication**, **Confidentiality**, **Integrity**
  - However, also raises *perimeter issues*
    (Try it yourself at http://www.net.berkeley.edu/vpn/)
## Application Proxies

- Can more directly control applications by requiring them to go through a proxy for external access
  - Proxy doesn’t simply forward, but acts as an application-level middleman

- Example: SSH gateway
  - Require all SSH in/out of site to go through gateway
  - Gateway logs authentication, **inspects decrypted text**
  - Site’s firewall configured to *prohibit any other SSH access*
SSH Gateway Example

host-to-gateway SSH session

gateway-to-remote host SSH session

Firewall

allow <port=22, host=1.3.5.7>
drop <port=22>
Application Proxies

• Can more directly control applications by requiring them to go through a proxy for external access
  – Proxy doesn’t simply forward, but acts as an application-level middleman

• Example: SSH gateway
  – Require all SSH in/out of site to go through gateway
  – Gateway logs authentication, inspects decrypted text
  – Site’s firewall configured to prohibit any other SSH access

• Provides a powerful degree of monitoring/control

• Costs?
  – Need to run extra server(s) per app (possible bottleneck)
  – Each server requires careful hardening
Why Have Firewalls Been Successful?

- **Central control** – *easy administration and update*
  - Single point of control: update one config to change security policies
  - Potentially allows rapid response
- **Easy to deploy** – *transparent to end users*
  - Easy incremental/total deployment to protect 1,000’s
- **Addresses an important problem**
  - Security vulnerabilities in network services are rampant
  - Easier to use firewall than to directly secure code …
Firewall Disadvantages?

• *Functionality loss – less connectivity, less risk*
  – May reduce network’s usefulness
  – Some applications don’t work with firewalls
    • Two peer-to-peer users behind different firewalls

• *The malicious insider problem*
  – Assume insiders are trusted
    • Malicious insider (or anyone gaining control of internal machine) can wreak havoc

• Firewalls establish a *security perimeter*
  – Like *Eskimo Pies*: “hard crunchy exterior, soft creamy center”
  – Threat from travelers with laptops, cell phones, …