CS 261 - Network Security
Lecture Notes Oct. 9th, 2008
Scribe: Hisham Zarka

TCP/IP Over Wireless Networks

Two months ago a security researcher announced a tool that would hijack sessions over a wireless network. How could you build such a tool?

- TCP Hijacking
- ARP spoofing: could inject arbitrary
- Fake AP

These techniques can allow an attacker to make himself a man-in-the-middle on a TCP/IP connection. Once the attacker is man-in-the-middle on a TCP connection, many higher-level protocols (such as HTTP) can subverted

On a wireless network, we can arbitrarily eavesdrop and inject packets. How can we use this to take control of a session? Using TCP sequence numbers.

Every TCP packet sent includes a sequence number of 32 bits which is incremented throughout the connection.

- If the sequence number is just right, the packet is received
- If the sequence number is too low, the packet is ignored
- If the sequence number is too high, the packet is buffered until an earlier packet with the right sequence number arrives

As an attacker on a wireless network, you can eavesdrop and determine what the next sequence number should be. Injecting a new packet with the correct sequence number is consequently trivial.

The attack

- The attacker injects a spoofed packet with the correct TCP sequence number (as observed) into the connection to the victim server
- The victim server receives the injected packet, and responds with an ACK acknowledging the updated TCP sequence number.
- The victim client receives the ACK with the updated TCP sequence number, which does not match its record of the TCP sequence number. It sends an ACK back to the victim server with the old TCP sequence number
- The victim server receives the ACK with the old TCP sequence number, and replies with another ACK containing the new TCP sequence number
- The victim client can no longer communicate with the victim server because the TCP sequence numbers are out-of-sync, but the attacker can communicate bidirectionally with the victim server (posing as the victim client)

- As a consequence of the above procedure, the victim client and victim server continue exchanging ACK packets in what is known as an ACK storm. There are methods to resynchronize the client and server to avoid the ACK storm

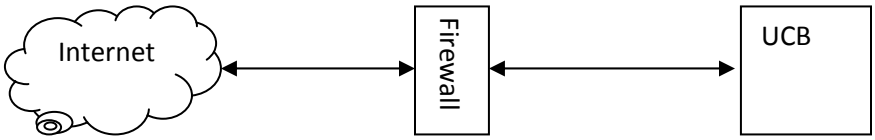
HTTP Hijacking

Many websites operate primarily over HTTP but redirect to an HTTPS site for login. This has a few limitations and is not completely secure:

- If the link to HTTPS login is served over an HTTP page, a man-in-the-middle attacker can simply remove the HTTPS from the link to the login page and intercept the traffic
- If there is an initial encrypted period followed by an unencrypted session (Gmail) the attacker can simply hijack the unencrypted session

Firewalls

Assume the following network topology with rules which can be specified on the firewall to inspect and drop packets



In this instance, the UCB network is connected to and protected from the internet by the firewall. What if we wanted to allow

What if we wanted to prevent incoming connections to port 139?

drop *:* → *:139

- Drops ALL packets destined to port 139, whether the packets are incoming or outgoing

drop *:*/world → *:139/ucb

- This rule will now only drop incoming packets destined to port 139
- Does not work for TCP because connections require packets to flow the other direction as well

drop *:*/world → *:139/ucb and SYN and not ACK

- What we really meant to say was: only block incoming connection requests, and this rule more accurately captures that sentiment
- But this makes an assumption about TCP stacks on internal hosts: the only way to establish a connection is with a packet containing a SYN but not ACK
- Effectively a blacklist

allow *:*/world → *:139/ucb and ACK
drop *:*/world → *:139/ucb

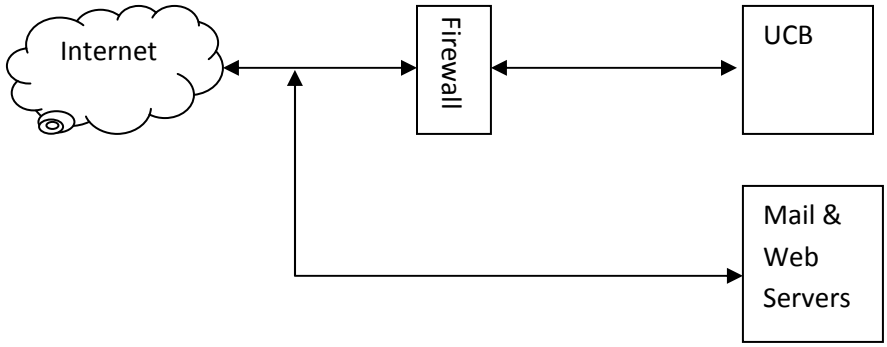
- This rule will allow incoming packets to port 139 only if the ACK bit is set
- Assumption about internal TCP stacks: if you get an ACK packet without an established connection, it will be dropped

allow *:*/world → *:*/ucb and ACK
drop *:*/world → *:*/ucb

- Will drop all incoming connections, but allow all outgoing connections

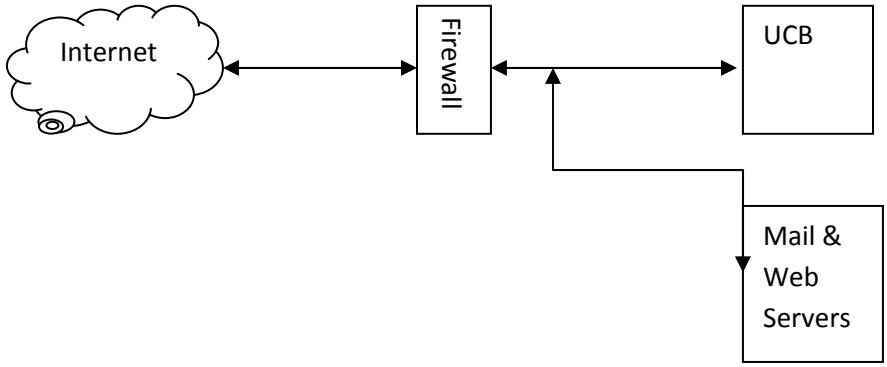
What if you want to add a mail and HTTP, accessible externally?

Put them outside the network?



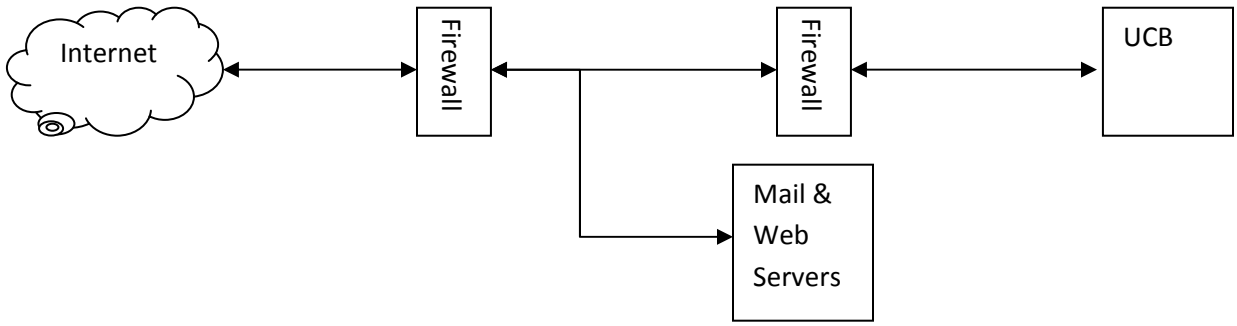
- They won't be protected at all

Put them in the internal network and poke a hole in the firewall



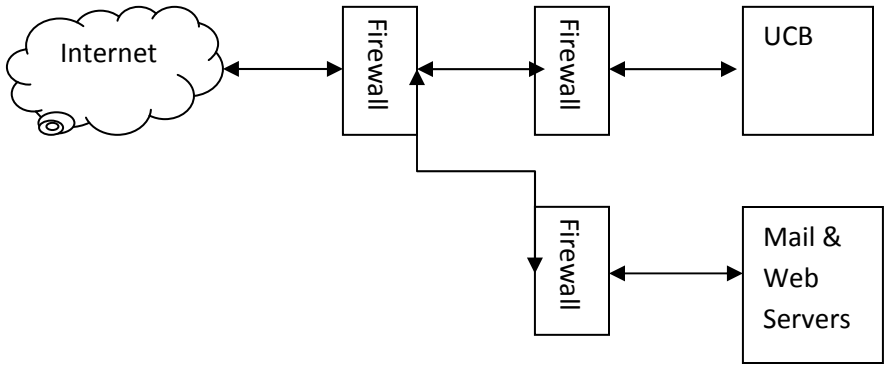
- But if the servers are compromised, the entire internal network is at risk

Put them on an internal network and then put other hosts behind yet another firewall



- If one of the servers is compromised, it may be able to listen on connections or hijack TCP sessions

Have two separate internal networks: one for completely protected hosts, and another for the servers



- Works pretty well, but if the servers need to access some of the protected hosts a more complicated topology is required

Reference Monitors

We have seen two instances of *reference monitors*:

- network firewalls which mediate network traffic
- OS-level code which approves or disapproves system calls

A reference monitor is a general system mediating access to a resource which satisfies three conditions:

- The reference monitor is *always invoked* (complete mediation)
- The reference monitor *protects itself* (can't attack/subvert reference monitor)
- The reference monitor is *simple* enough to have assurance

Are firewalls always invoked? In a simple topology: yes, except:

- ***VPNs***: the tunnel may be allowed, but then the connections forwarded by VPN maybe should not have been allowed and will bypass the firewall
- ***Wireless access points***: can allow external access to the inside of a computer network
- ***Laptop sneakernet***: a user's laptop may have been infected at a conference, and then brought physically to the inside of the network