Network security

- First need to understand basic networking
  - Network Architecture
  - IP
  - UDP
  - TCP
  - DNS
- And vulnerabilities in their architecture

Packet Switched Network Architecture

- Network nodes
  - Hosts (individual computers)
  - Routers/Switches (specialized devices that forward messages along)
- Links
  - Transmission medium between nodes
- Packets
  - Self-identifying encapsulated messages sent between nodes across links
- Protocol
  - Particular implementation of a network service (i.e. TCP implements reliable stream communication)

Protocols are structured in layers

- Each layer provides a given service and only depends on the layer below it (in principal)
- Physical: sending bits (e.g. 2.4Ghz OFDM RF)
- Data link: framing (e.g. Ethernet)
- Network: end-to-end packet delivery (e.g. IP)
- Transport: reliable message delivery, flow/congestion control (e.g. TCP)
- …
- Application layer: the application (e.g. SMTP, HTTP)

Layer encapsulation via packet headers

TCP/IP Protocol Stack

- Application protocol (e.g. HTTP)
  - Application
  - Transport
- Network
  - IP protocol
  - IP protocol
- Link
  - Data Link
  - Network Access
  - Data Link
Data Formats

Layering by example...

- ROUGHLY, what happens when I click on a Web page from UCSD?

Application layer (HTTP)

- Turn click into HTTP request

Transport layer (TCP)

- Break message into packets (TCP segments)
- Should be delivered reliably & in-order

Network layer: IP Addressing

- Address each packet so it can traverse network and arrive at host
- Addresses are generally globally unique

Application layer?
Name resolution (DNS)

- Where is www.yahoo.com?

GET http://www.yahoo.com/r/mp HTTP/1.1
Host: www.yahoo.com
Connection:keep-alive

GET http://www.yahoo.com/r/mp HTTP/1.1
Host: www.yahoo.com
Connection:keep-alive
**Network layer: IP Routing**

- Each router forwards packet towards destination

**Datalink layer (Ethernet)**

- Break message into frames
- Media Access Control (MAC)
  - Can I send now? Can I send now?
- Send frame

**Physical layer**

- 2.4GHz Radio
- 802.11b Wireless Access Point
- DS/FH Radio (1-11Mbps)
- Cat5 Cable (4 wires) 100BaseTX Ethernet 100Mbps
- 62.5/125um 850nm MMF 1000BaseSX Ethernet 1000Mbps
- 100Base TX Ethernet 100Mbps
- 1000BaseSX Ethernet 1000Mbps

**Data Formats**

- Application message - data
- TCP
- IP
- Ethernet
- Frame (data)
- Segment (TCP)
- Packet (IP)
- Application message - data

**Today’s IP Packet Header**

```
<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver</td>
<td>HL</td>
<td>TOS</td>
<td>length</td>
</tr>
<tr>
<td>source address</td>
<td>destination address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>options (if any)</td>
<td>data (if any)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**IP Protocol Functions (Summary)**

- Routing
  - IP host knows location of local router (gateway)
  - IP gateway must know route to other networks
    - Packets usually take multiple hops to get to destination
  - Addresses are globally meaningful
    - 32 bits (IPv4), address separated into network part and host part
- Error reporting
  - ICMP packet to source if packet is dropped
- Fragmentation and reassembly
  - If max-packet-size less than the user-data-size
  - TTL field: decremented after every hop
    - Packet dropped if TTL=0. Prevents infinite loops.
**Fragmentation**

- Sender writes unique value in identification field.
- If router fragments packet it copies id into each fragment.
- Offset field indicates position of fragment in bytes (offset 0 is first). 
  - MoreFragments flag indicates that this isn’t the last fragment.
  - Don’tFragment flag tells gateway not to fragment.
- All routers must support 576 byte packets (MTU).

---

**IP Fragmentation and Reassembly**

One large datagram becomes several smaller datagrams.

---

**UDP**

**User Datagram Protocol**

- Unreliable transport on top of IP:
  - Ports identify individual endpoint/process.
  - No acknowledgments, congestion control, message continuation.

---

**Transmission Control Protocol (TCP)**

- Reliable bi-directional byte stream between processes.
- Connection-oriented:
  - Conversation between two endpoints with a beginning and an end.
- Flow control:
  - Prevents sender from over-running receiver buffers.
- Congestion control:
  - Prevents sender from over-running network buffers.

---

**TCP Delivery**

- Each byte in a packet has its own sequence number (e.g. byte 1 is sequence number 26).
- When the receiver gets a packet it sends an acknowledgement indicating the next sequence number it expects (e.g. “ack 27” if the previous data packet was only a single byte).
- If the sender doesn’t get an ACK after a while it retransmits (there are some optimizations here).
- Eventually, all data is delivered in order or the connection fails.
### TCP Header Format

- **Ports** plus IP addresses identify a connection

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>16</td>
</tr>
<tr>
<td>DstPort</td>
<td>16</td>
</tr>
<tr>
<td>SeqNum</td>
<td>32</td>
</tr>
<tr>
<td>AckNum</td>
<td>32</td>
</tr>
<tr>
<td>HdrLen</td>
<td>4</td>
</tr>
<tr>
<td>Flags</td>
<td>13</td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td>16</td>
</tr>
<tr>
<td>URGPr</td>
<td>1</td>
</tr>
<tr>
<td>Options</td>
<td>(variable)</td>
</tr>
<tr>
<td>Data</td>
<td>(variable)</td>
</tr>
</tbody>
</table>

### Connection Setup:

**Agree on initial Sequence #’s**

- Three-way handshake

  - Active participant (client)
  - Passive participant (server)

```
SYN, SequenceNum = x
SYN + ACK, SequenceNum = y, Acknowledgment = y + 1
ACK, Acknowledgment = x + 1 + data
```

### Basic TCP/IP Security Issues

- **No Authentication/Authorization**
  - Anyone can send to any port on any host; port scanning, Denial of Service (DoS), worms
- **No Attribution**
  - Nothing enforces correctness of IP address; IP spoofing
- **Network packets not private**
  - Intermediate networks not necessarily trusted; packet sniffing
- **TCP/IP state can be easy to guess**
  - TCP connection spoofing, blind port scanning

### 1. Packet Sniffing

- Promiscuous NIC reads all packets
  - Read all unencrypted data (e.g., “wireshark”)
  - ftp, telnet (and POP, IMAP) send passwords in clear!

```
Alice -> Network -> Eve -> Bob
```

**Prevention:** Encryption

### 2. TCP Connection Spoofing

- Why random initial sequence numbers? \( (SN_C, SN_S) \)
- Suppose init. sequence numbers are predictable
  - Attacker can create TCP session on behalf of forged source IP
    - Breaks IP-based authentication (e.g. SPF, /etc/hosts)

```
attacker
\[ SYN, srcIP=victim, \]
\[ ACK, srcIP=victim, AN=predicted SN, \]
\[ command, \]
```

```
Server
\[ SYN/ACK, dstIP=victim, SN=server SN, \]
```

```
Vicitm
\[ server thinks command is from victim IP addr \]
```

### Example DoS vulnerability

- Suppose attacker can guess seq. number for an existing connection:
  - Attacker can send Reset packet to close connection. Results in DoS.
  - Naively, success prob. is \( 1/2^{32} \) (32-bit seq. #’s).
  - Most systems allow for a large window of acceptable seq. #’s
    - Much higher success probability.

- Attack is most effective against long lived connections (expensive to set up again; BGP)
Random initial TCP SNs

- Unpredictable SNs prevent basic packet injection
  - ... but attacker can inject packets after eavesdropping to obtain current SN
- Most TCP stacks now generate random SNs
  - Random generator should be unpredictable
- GPR’06: Linux RNG for generating SNs is predictable
  - Attacker repeatedly connects to server
  - Obtains sequence of SNs
  - Can predict next SN
  - Attacker can now do TCP spoofing
    (create TCP session with forged source IP)

Blind port scanning

- Similar issue: identification field
  - Hosts typically increment by one after each packet to ensure id field is unique (recall: for fragmentation)
  - If you receive a pkt from host A at time t1 with id = 10, and another packet at time t2 with id = 12, you can infer... that host A sent another packet somewhere

Protocol Vulnerabilities

- Network protocols are frequently designed assuming all actors are benign
- What if one participant in a communication doesn’ t obey the protocol?
  - Simple example: routing
    - Nothing prevents UCSD from claiming to “own” 128.95/16 (University of Washington)
    - This happens, both on purpose and by accident
  - Other examples
    - TCP congestion control
    - DNS poisoning

How TCP congestion control works

- Sender maintains “congestion window”
  - Limit on amount of outstanding data
  - Grows when data is successfully delivered
  - Shrinks when data is lost
- Receiver sends ACKs in response to data
  - ACKs tell sender that data has been received
  - Indicate the next data item expected
- Works if everyone plays fair
  - Sender could ignore protocol and send faster
  - What about receiver? (e.g., Web browser)

Sources of vulnerability

- ACKs mean things that they don’ t prove
  - I was sent in response to a data packet
  - That data packet has been received
  - I have received all the data up to X-1
  - I have (still) not yet received data X
- Sender assumes things that aren’ t necessarily true
  - At most one ACK generated per data packet
  - Every ACK acknowledges a full-sized packet
**What’s supposed to happen**

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-Trip Time (RTT)</td>
<td></td>
</tr>
<tr>
<td>Data 1:1460</td>
<td>ACK 1461</td>
</tr>
<tr>
<td>Data 1461:2920</td>
<td>Data 2921:4380</td>
</tr>
<tr>
<td>ACK 2921</td>
<td>ACK 4381</td>
</tr>
</tbody>
</table>

- Rule: grow window by one full-sized packet for each valid ACK received
- Congestion window doubles each round trip time

**Example of breaking the rules**

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-Trip Time (RTT)</td>
<td></td>
</tr>
<tr>
<td>Data 1:1461</td>
<td>ACK 486</td>
</tr>
<tr>
<td>Data 486:973</td>
<td>ACK 973</td>
</tr>
<tr>
<td>ACK 973</td>
<td>ACK 1461</td>
</tr>
<tr>
<td>Data 1461:2921</td>
<td>Data 2921:4381</td>
</tr>
<tr>
<td>Data 4381:5841</td>
<td>Data 5841:7301</td>
</tr>
</tbody>
</table>

- Rule: grow window by one full-sized packet for each valid ACK received
- Send \( M \) ACKs for one pkt
- Growth factor proportional to \( M \)

**Why my Web browser is faster than yours**

- Modified Client
- Normal Client

![Graph showing page fetch from CNN.com](image)

**Domain Name System**

- Hierarchical Name Space
  - root
  - edu
  - com
  - uk
  - ca
  - org
  - net
  - wisc
  - ucb
  - ucsd
  - cimc
  - mit
  - cs
  - ece
  - www

**DNS Root Name Servers**

- Hierarchical service
  - Root name servers for top-level domains
  - Authoritative name servers for subdomains
  - Local name resolvers contact authoritative servers when they do not know a name

**DNS Lookup Example**

- DNS record types (partial list):
  - **NS**: name server (points to other server)
  - **A**: address record (contains IP address)
  - **MX**: address in charge of handling email
  - **TXT**: generic text (e.g., used to distribute site public keys (DKIM))
Caching

- DNS responses are cached
  - Quick response for repeated translations
  - Useful for finding servers as well as addresses
    - NS records for domains

- DNS negative queries are cached
  - Save time for nonexistent sites, e.g. misspelling

- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record

DNS Packet

- Query ID:
  - 16 bit random value
  - Links response to query
  - NS records for domains
  - DNS negative queries are cached
  - Save time for nonexistent sites, e.g. misspelling
  - Cached data periodically times out
    - Lifetime (TTL) of data controlled by owner of data
    - TTL passed with every record

Resolver to NS request

- Query ID: 16 bit random value
  - Links response to query

Response to resolver

- Response contains IP addr of next NS server (called "glue")
- Response ignored if unrecognized QueryID

Authoritative response to resolver

- Bailiwick checking: response is cached if it is within the same domain of query
  (i.e. a.com cannot set NS for b.com)
- Final answer

Basic DNS Vulnerabilities

- Users/hosts trust the host-address mapping provided by DNS:
  - Used as basis for many security policies: Browser "same origin" policy, URL address bar, user trust

- Obvious problems
  - Interception of requests or compromise of DNS servers can result in incorrect or malicious responses
    - e.g. hijack network route to spoof DNS
  - Solution – authenticated requests/responses
    - Provided by DNSSEC … but no one uses DNSsec
### DNS cache poisoning (a la Kaminsky '08)

- Victim machine visits attacker's website, downloads Javascript

#### Diagram:
- User browser connects to `a.bank.com`
- Local DNS resolver queries `a.bank.com`
- Query with ID `x_1`
- 256 responses:
  - Random QID `y_1, y_2, ..., y_{256}`
  - `NS bank.com=ns.bank.com`
  - `A ns.bank.com=attackerIP`
- Attacker wins if \( \exists j: x_1 = y_j \)
- Response is cached and attacker owns `bank.com`

### If at first you don't succeed ...

- Victim machine visits attacker's website, downloads Javascript

#### Diagram:
- User browser connects to `b.bank.com`
- Local DNS resolver queries `b.bank.com`
- Query with ID `x_2`
- 256 responses:
  - Random QID `y_1, y_2, ..., y_{256}`
  - `NS bank.com=ns.bank.com`
  - `A ns.bank.com=attackerIP`
- Attacker wins if \( \exists j: x_2 = y_j \)
- Response is cached and attacker owns `bank.com`

### Defenses

- Increase Query ID size. How? Some proposals
  - Randomize src port, additional 11 bits
    - Now attack takes several hours
  - Ask every DNS query twice:
    - Attacker has to guess QueryID correctly twice (32 bits)
    - Not clear DNS system can handle load

### Summary

- Core protocols not designed for security
  - Eavesdropping, Packet injection, Route stealing, DNS poisoning
  - Patched over time to prevent basic attacks
    - (e.g. random TCP SN)
  - More secure variants exist (limited deployment)
    - IP -> IPsec
    - DNS -> DNSsec
    - BGP -> SBGP
- Still need to be careful about protocol semantics