Lecture 25
November 29, 2023

Proxy

- Intermediate agent or server acting on behalf of endpoint without allowing a direct connection between the two endpoints
  - So each endpoint talks to proxy, thinking it is talking to other endpoint
  - Proxy decides whether to forward messages, and whether to alter them

Proxy Firewall

- Access control done with proxies
  - Usually bases access control on content as well as source, destination addresses, etc.
  - Also called an *applications level* or *application level firewall*
  - Example: virus checking in electronic mail
    - Incoming mail goes to proxy firewall
    - Proxy firewall receives mail, scans it
    - If no virus, mail forwarded to destination
    - If virus, mail rejected or disinfected before forwarding

Example

- Want to scan incoming email for malware
- Firewall acts as recipient, gets packets making up message and reassembles the message
  - It then scans the message for malware
  - If none, message forwarded
  - If some found, mail is discarded (or some other appropriate action)
- As email reassembled at firewall by a mail agent acting on behalf of mail agent at destination, it’s a proxy firewall (application layer firewall)

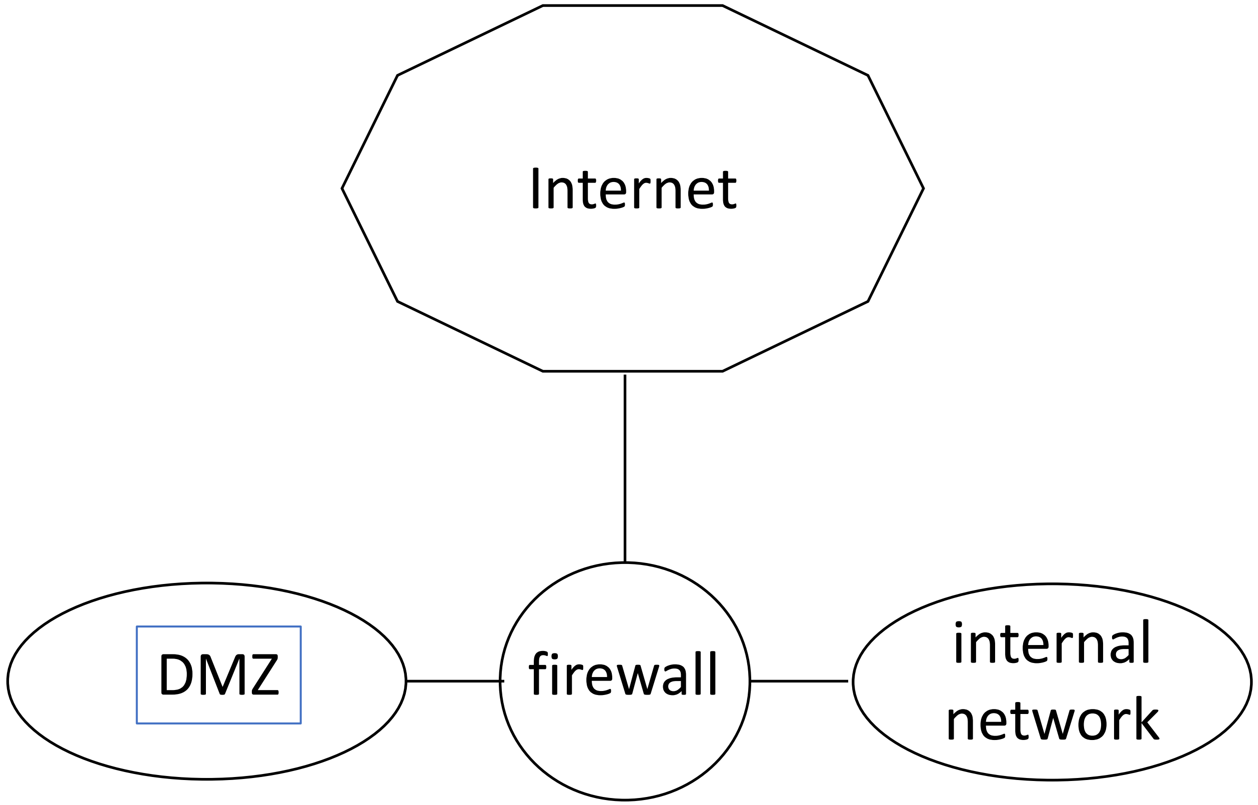
Stateful Firewall

- Keeps track of the state of each connection
- Similar to a proxy firewall
  - No proxies involved, but this can examine contents of connections
  - Analyzes each packet, keeps track of state
  - When state indicates an attack, connection blocked or some other appropriate action taken

Network Organization: DMZ

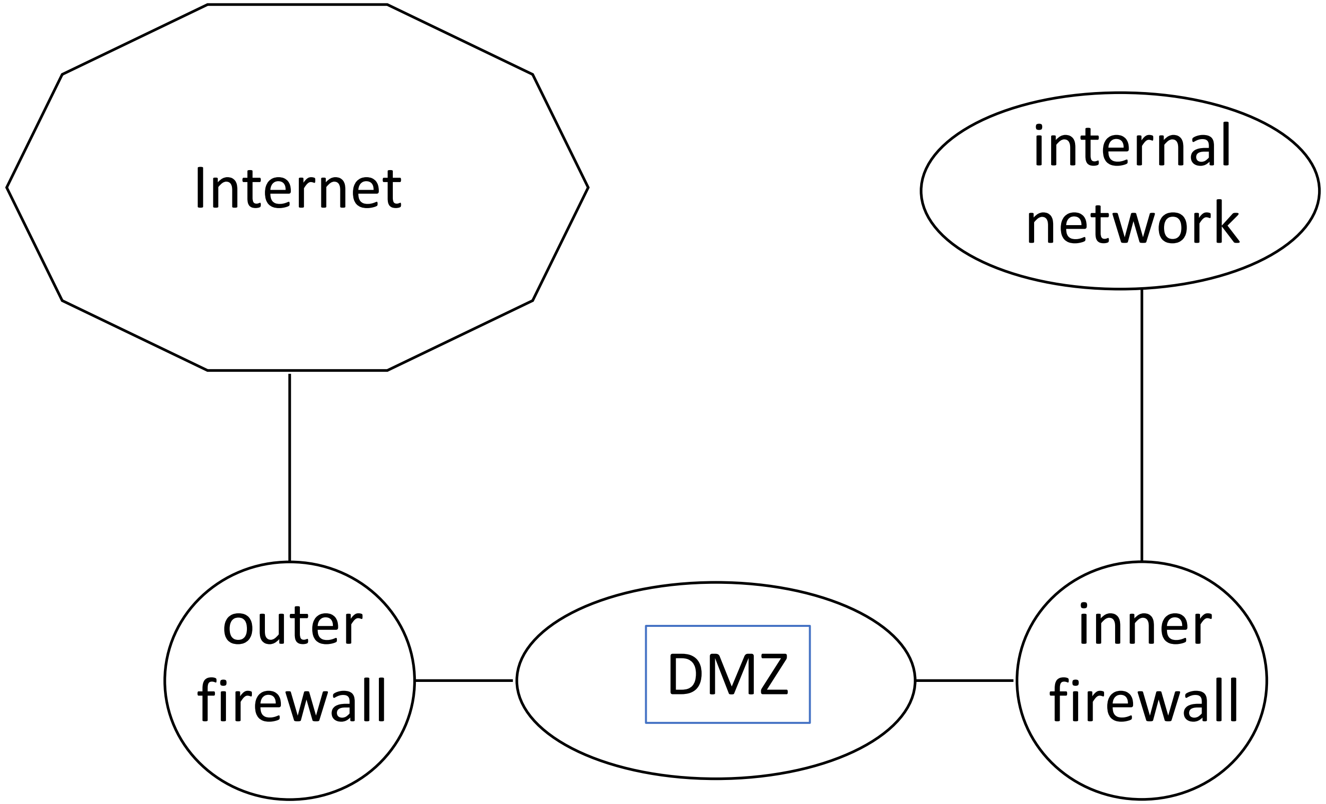
- DMZ is portion of network separating a purely internal network from external network
- Usually put systems that need to connect to the Internet here
- Firewall separates DMZ from purely internal network
- Firewall controls what information is allowed to flow through it
    - Control is bidirectional; it control flow in both directions

One Setup of DMZ



One dual-homed firewall that routes messages to internal network or DMZ as appropriate

Another Setup of DMZ



Two firewalls, one (outer firewall) connected to the Internet, the other (inner firewall) connected to internal network, and the DMZ is between the firewalls

Anonymity on the Web

- Recipients can determine origin of incoming packet
  - Sometimes not desirable
- Anonymizer: a site that hides origins of connections
  - Usually a proxy server
    - User connects to anonymizer, tells it destination
    - Anonymizer makes connection, sends traffic in both directions
  - Destination host sees only anonymizer

Example: *anon.penet.fi*

Offered anonymous email service

- Sender sends letter to it, naming another destination
- Anonymizer strips headers, forwards message
    - Assigns an ID (say, 1234) to sender, records real sender and ID in database
    - Letter delivered as if from anon1234@anon.penet.fi
- Recipient replies to that address
    - Anonymizer strips headers, forwards message as indicated by database entry

Problem

- Anonymizer knows who sender, recipient *really* are
- Called *pseudo-anonymous remailer* or *pseudonymous remailer*
  - Keeps mappings of anonymous identities and associated identities
- If you can get the mappings, you can figure out who sent what

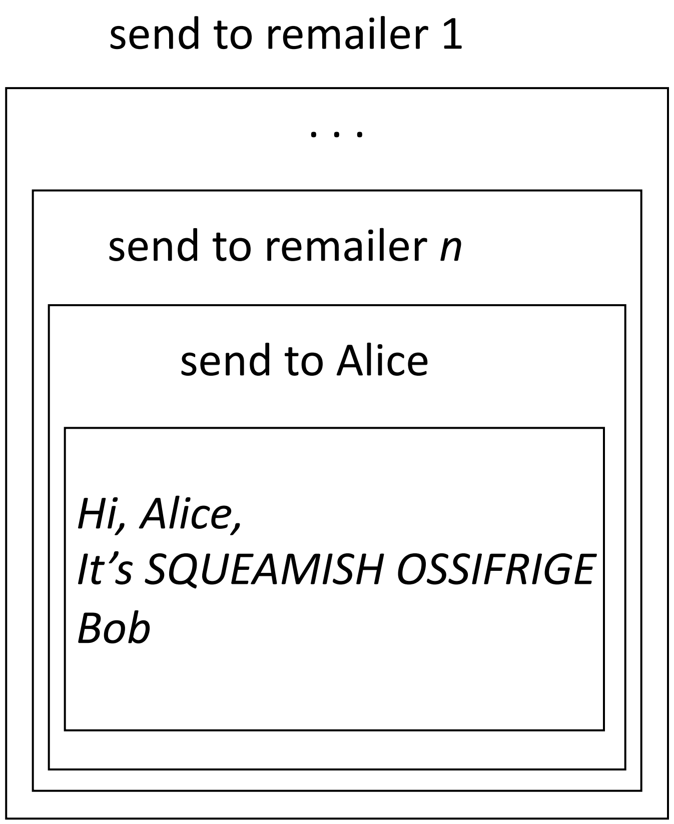
More *anon.penet.fi*

- Material claimed to be copyrighted sent through site
- Finnish court directed owner to reveal mapping so plaintiffs could determine sender
- Owner appealed, lost, subsequently shut down site

Cypherpunk Remailer

- Remailer that deletes header of incoming message, forwards body to destination
- Also called *Type I Remailer*
- No record kept of association between sender address, remailer's user name
    - Prevents tracing, as happened with *anon.penet.fi*
- Usually used in a chain, to obfuscate trail
    - For privacy, body of message may be enciphered

Cypherpunk Remailer Message



- Encipher message
- Add recipient address
- Encipher and add remailer *n*'s address
- . . .
- Encipher and add remailer 1's address
- Send this to remailer 1

Weaknesses

- Attacker monitoring entire network
  - Observes in, out flows of remailers
  - Goal is to associate incoming, outgoing messages
- If messages are cleartext, trivial
  - So assume all messages enciphered
- So use traffic analysis!
  - Used to determine information based simply on movement of messages (traffic) around the network

Attacks

- If remailer forwards message before next message arrives, attacker can match them up
    - Hold messages for some period of time, greater than the message interarrival time
    - Randomize order of sending messages, waiting until at least *n* messages are ready to be forwarded
        - Note: attacker can force this by sending *n*–1 messages into queue

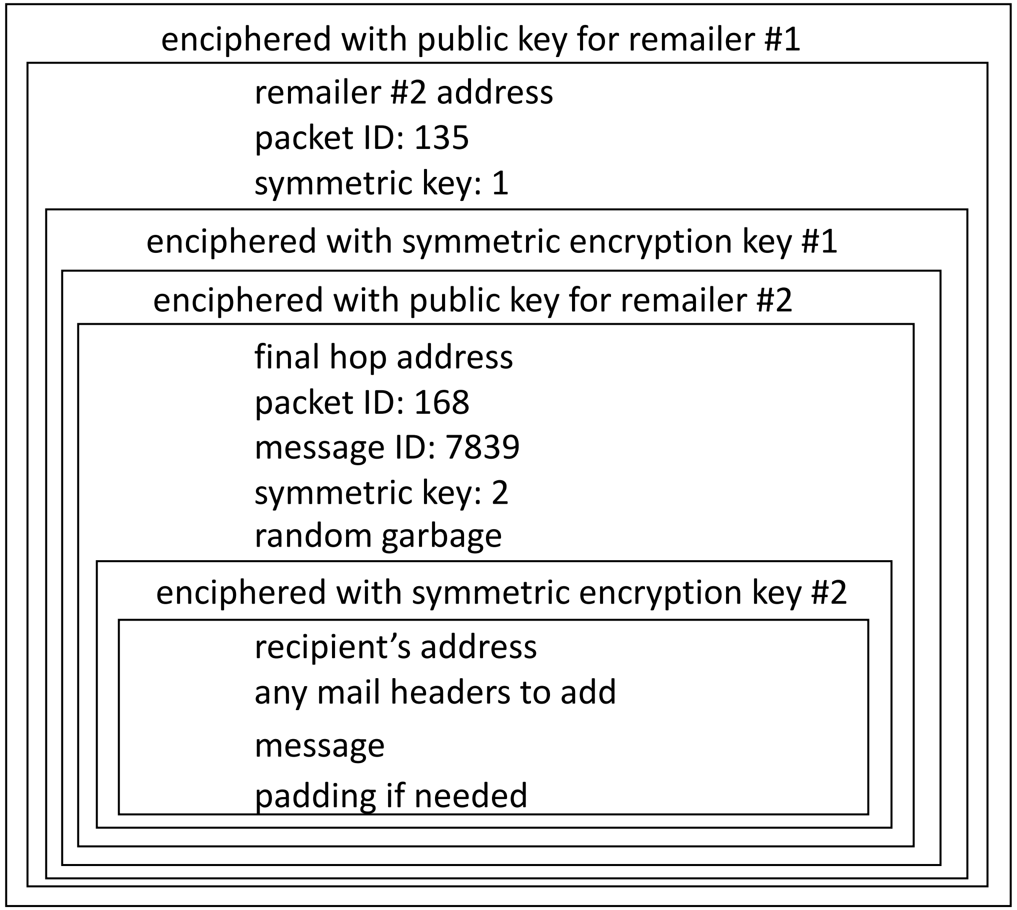
Attacks

- As messages forwarded, headers stripped so message size decreases
  - Pad message with garbage at each step, instructing next remailer to discard it
- Replay message, watch for spikes in outgoing traffic
  - Remailer can’t forward same message more than once

Mixmaster (Cypherpunk Type 2) Remailer

- Cypherpunk remailer that handles only enciphered mail and pads (or fragments) messages to fixed size before sending them
  - Also called Type 2 Remailer
  - Designed to hinder attacks on Cypherpunk remailers
    - Messages uniquely numbered
    - Fragments reassembled *only* at last remailer for sending to recipient

Cypherpunk Remailer Message



Onion Routing

- Method of routing so each node in the route knows only the previous and following node
  - Typically, first node selects the route
  - Intermediate node may be able to change rest of route
- Each intermediate node has public, private key pair
  - Public key available to all nodes and any proxies
- Client, server have proxies to handle onion routing

Heart of the Onion Route

{ $expires$ || $nexthop$ || E_F || k_F || E_B || k_B || $payload$ } pub_r

- *payload*: data associated with message
- *expires*: expiration time for which *payload* is to be saved
- *nexthop*: node to forward message to
- pub_r: public key of next hop (node)
- E_F, k_F : encryption algorithm, key to be used when sending message forward to server
- E_B, k_B : encryption algorithm, key to be used when sending message backwards to client

Notes About the Heart

- *payload* may itself be a message of this form or the data being sent
- Each router has table storing:
  - Virtual circuit number associated with a route
  - E_F, k_F, E_B, k_B for the next, previous nodes on the route
  - Next router to which messages using this route are to be forwarded
    - If last router on route, this is NULL (as is *nexthop* in the packet)

Creating a Route

- Client’s proxy determines route for the message
  - Can be defined exactly, or loosely, where the intermediate routers can route messages to next hop over other routes
- Create onion encapsulating route, put it in a *create* message and add virtual circuit number
- Forward to next (second) router on path
- That router deciphers the onion using its private key (“peeling the onion”)
  - Compare it to what’s in table; if replay, discard

Creating a Route

- Router creates new virtual circuit number, and add to table:
  - (virtual circuit number in message, created virtual circuit number) pair
  - Keys, algorithms in onion
- Router generates new *create* message, puts assigned virtual circuit number and “peeled” onion in it
  - This is smaller than the onion received, so add padding to make it the same size
- Forward it to next hop

Sending a Message

- Sender applies decryption algorithms corresponding to each backwards encryption algorithm along the route
- Example: route begins at *W*, then through *X* and *Y* to *Z*; *W* constructs this:

$$d_X(k_X, d_Y(k_Y, d_Z(k_Z, m)))$$

- Sends this to *X*, which uses its E_B to encrypt message, getting

$$d_Y(k_Y, d_Z(k_Z, m))$$

- Forwards this to *Y*, which uses its E_B to encrypt message, getting

$$d_Z(k_Z, m)$$

- Forwards this to *Z*, which uses its E_B to encrypt message, getting *m*

Potential Attacks

- If client's proxy compromised, attacker can see all routes selected and all messages, and so may be able to deduce server
- If server's proxy compromised, attacker can see all messages but cannot deduce the routes
- If router compromised, attacker can determine only the previous, next routers in path
  - In particular, the attacker cannot read the encrypted onion
- Attacker can see all traffic on network
  - Matching client, server message sizes; that's why all messages are padded to same size
  - Observing the flow of messages; have the onion network send meaningless messages to obscure that flow

Example: Tor (The Onion Router)

- Connects clients, servers over virtual circuits set up among onion routers (*OR*)
  - Each OR has identity key, onion key
  - Identity key signs information about router
  - Onion key used to read requests to set up circuits; changed periodically
  - All virtual circuits over TLS, and a third TLS key established for this
- Basic message unit: *cell*, always 512 bytes long
  - Control cell: header contains command directing recipient to do something
    - Create a circuit, circuit created, destroy a circuit
  - Relay cell: deals with an established circuit
    - Open stream, stream opened, extend circuit, circuit extended, close stream cleanly, close broken stream, cell contains data

Setting Up Virtual Circuit

- Set up over TLS connections
  - Several circuits may use same TLS connection to reduce overhead
- Streams move data over virtual circuits
  - Several streams may be multiplexed over one circuit
- Client's onion proxy OP_c needs to know where ORs are
  - Tor uses directory services for this; group of well-known ORs track information about usable ORs, including keys, addresses
  - OP_c contacts one such directory server, gets information from it, chooses path

Setting Up Virtual Circuit

- Tor uses 3 ORs (OR_1, OR_2, OR_3); client, server proxies OP_c, OP_s
- $RSA(x)$ is enciphering of message x using onion key of destination OR
- g, p as in Diffie-Hellman
- $x_1, \ldots, x_n$ and $y_1, \ldots\ y_n$ generated randomly; $k_i = g^{x_i y_i} \bmod p$, and forward, backwards keys selected from this
- $h(x)$ cryptographic hash of x
- All links are over TLS and so encrypted (TLS keys not shown on next slide)

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_1:

$$OP_c \xrightarrow{\{\, create \,||\, c_1 \,||\, RSA(g^{x_1}) \,\}} OR_1$$

$$OP_c \xleftarrow{\{\, created \,||\, c_1 \,||\, g^{y_1} \,||\, h(k_1) \,\}} OR_1$$

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_2:

$$OP_c \xrightarrow{\{\, relay \,||\, c_1 \,||\, \{\, extend \,||\, OR_2 \,||\, RSA(g^{x_2}) \,\}\,\}} OR_1$$

$$OR_1 \xrightarrow{\{\, create \,||\, c_2 \,||\, RSA(g^{x_2}) \,\}} OR_2$$

$$OR_1 \xleftarrow{\{\, created \,||\, c_2 \,||\, g^{y_2} \,||\, h(k_2) \,\}} OR_2$$

$$OP_c \xleftarrow{\{\, relay \,||\, c_1 \,||\, \{\, extended \,||\, g^{y2} \,||\, h(k_2) \,\}} OR_1$$

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_3:

$$OP_c \xrightarrow{\{\, relay \;||\; c_1 \;||\; \{\, extend \;||\; OR_3 \;||\; RSA(g^{x_3}) \,\}\,\}} OR_1$$

$$OR_1 \xrightarrow{\{\, relay \;||\; c_2 \;||\; \{\, extend \;||\; OR_3 \;||\; RSA(g^{x_3}) \,\}\,\}} OR_2$$

$$OR_2 \xrightarrow{\{\, create \;||\; c_3 \;||\; RSA(g^{x_3}) \,\}} OR_3$$

$$OR_3 \xleftarrow{\{\, created \;||\; c_3 \;||\; g^{y_3} \;||\; h(k_3) \,\}} OR_3$$

$$OR_1 \xleftarrow{\{\, relay \;||\; c_2 \;||\; \{\, extended \;||\; g^{y_3} \;||\; h(k_3) \,\}\,\}} OR_2$$

$$OR_1 \xleftarrow{\{\, relay \;||\; c_1 \;||\; \{\, extended \;||\; g^{y_2} \;||\; h(k_2) \,\}\,\}} OP_c$$

After All This . . .

- OP_c has forward keys for OR_1, OR_2, OR_3; call them f_1, f_2, f_3
  - Here, $f_i = g^{y_i} \bmod p$
- To send message m to server, client sends m to OP_c
  - OP_c enciphers it using AES-128 in counter mode, getting $\{ \{ \{ m \}f_3 \}f_2 \}f_1$
  - It puts this into a relay cell and sends it to OR_1
- OR_1 deciphers cell, determines next hop by looking up virtual circuit number in its table, puts $\{ \{ m \}f_3 \}f_2$ into another relay cell, forwards it to OR_2
- OR_2 does same, and forwards it to OR_3
- OR_3 deciphers cell, either does what m requests (eg, open TLS connection to server) or forwards payload m to server

Server Replies

- Server sends reply *r* to OR_3
- OR_3 enciphers it using its backwards key, embeds it in relay cell, forwards it to OR_2
- OR_2 uses circuit number to determine OR_1, enciphers cell using its backwards key, forwards it to OR_1
- OR_1 does same but forwards it to OP_c
- OP_c has all the forward keys, and so can decipher the message and forward it to client

Use Problems

Adversary wants to determine who is using onion routing network

- Attack: monitor the client, known entry router
  - Solution: use unlisted entry routers
  - Example: Tor uses *bridge relays* that are not listed in Tor directories; to find them, go to specific web page or email a specific set of addresses; result is a list of entry routers (bridges) that OP_c can use
- Attack: examine packets sent from a client looking for structures indicating that they are intended for onion routers
  - Solution: obfuscate packet contents; endpoint deobfuscates it
  - Example: Tor has *pluggable transports* that do this