Trust

• Goal of certificate: bind correct identity to DN
• Question: what is degree of assurance?
• X.509v4, certificate hierarchy
  • Depends on policy of CA issuing certificate
  • Depends on how well CA follows that policy
  • Depends on how easy the required authentication can be spoofed
• Really, estimate based on the above factors
Example: Passport Required

• DN has name on passport, number and issuer of passport
• What are points of trust?
  • Passport not forged and name on it not altered
  • Passport issued to person named in passport
  • Person presenting passport is person to whom it was issued
  • CA has checked passport and individual using passport
PGP Certificates

• Level of trust in signature field
• Four levels
  • Generic (no trust assertions made)
  • Persona (no verification)
  • Casual (some verification)
  • Positive (substantial verification)
• What do these mean?
  • Meaning not given by OpenPGP standard
  • Signer determines what level to use
  • Casual to one signer may be positive to another
Identity on the Web

• Host identity
  • Static identifiers: do not change over time
  • Dynamic identifiers: changes as a result of an event or the passing of time

• State and Cookies

• Anonymity
  • Anonymous email
  • Anonymity: good or bad?
Host Identity

• Bound up to networking
  • Not connected: pick any name
  • Connected: one or more names depending on interfaces, network structure, context

• *Name* identifies principal

• *Address* identifies location of principal
  • May be virtual location (network segment) as opposed to physical location (room 222)
Example

• Layered network
  • MAC layer
    • Ethernet address: 00:05:02:6B:A8:21
    • AppleTalk address: network 51, node 235
  • Network layer
    • IP address: 192.168.35.89
  • Transport layer
    • Host name: cherry.orchard.chekhov.ru
Danger!

• Attacker spoofs identity of another host
  • Protocols at, above the identity being spoofed will fail
  • They rely on spoofed, and hence faulty, information
• Example: spoof IP address, mapping between host names and IP addresses
Domain Name Server

- Maps transport identifiers (host names) to network identifiers (host addresses)
  - Forward records: host names $\rightarrow$ IP addresses
  - Reverse records: IP addresses $\rightarrow$ host names
- Weak authentication
  - Not cryptographically based
  - Various techniques used, such as reverse domain name lookup
Reverse Domain Name Lookup

• Validate identity of peer (host) name
  • Get IP address of peer
  • Get associated host name via DNS
  • Get IP addresses associated with host name from DNS
  • If first IP address in this set, accept name as correct; otherwise, reject as spoofed

• If DNS corrupted, this won’t work
Floating (Dynamic) Identifiers

• Assigned to principals for a limited time
  • Server maintains pool of identifiers
  • Client contacts server using *local identifier*
    • Only client, server need to know this identifier
  • Server sends client *global identifier*
    • Client uses global identifier in other contexts, for example to talk to other hosts
    • Server notifies intermediate hosts of new client, global identifier association
Example: DHCP

• DHCP server has pool of IP addresses
• Laptop sends DHCP server its MAC address, requests IP address
  • MAC address is local identifier
  • IP address is global identifier
• DHCP server sends unused IP address
  • Also notifies infrastructure systems of the association between laptop and IP address
• Laptop accepts IP address, uses that to communicate with hosts other than server
Example: Gateways

- Laptop wants to access host on another network
  - Laptop’s address is 10.1.3.241
- Gateway assigns legitimate address to internal address
  - Say IP address is 101.43.21.241
  - Gateway rewrites all outgoing, incoming packets appropriately
  - Invisible to both laptop, remote peer
- Internet protocol NAT works this way
Weak Authentication

• Static: host/name binding fixed
• Dynamic: host/name binding varies over time
  • Must update reverse records in DNS
    • Otherwise, the reverse lookup technique fails
  • Cannot rely on binding remaining fixed unless you know the period of time over which the binding persists
DNS Security Issues

• Trust is that name/IP address binding is correct
• Goal of attacker: associate incorrectly an IP address with a host name
  • Assume attacker controls name server, or can intercept queries and send responses
Attacks

• Change records on server
• Add extra record to response, giving incorrect name/IP address association
  • Called “cache poisoning”
• Attacker sends victim request that must be resolved by asking attacker
  • Attacker responds with answer plus two records for address spoofing (1 forward, 1 reverse)
  • Called “ask me”
DNS Security Extensions (DNSSEC)

- DNS organizes information into *resource records* (RRs)
  - CNAME RR: canonical name for host
- DNSSEC adds some RRs for cryptographic authentication of record
  - RRSIG RR: signature
    - These associate digital signature with sets of records in DNS
  - DNSKEY RR: public key associated with DNS server
    - Resolver uses this to verify signature sent with DNS records
- Resolver requests record corresponding to host name
  - Server responds with NSEC RR showing *next* valid host name in sorted order
  - NSEC RR: next host name (the one following the host these RRs refer to)
    - Tells querying host that queried-for host does not exist in that domain
NSEC RR Problem and Solution

• Attack: derive all host names in domain by sending queries for host names that have no corresponding addresses

• Solution: NSEC3 RR is like NSEC RR, but host name replaced by cryptographic hash of host name
  • Now attacker cannot get the host names of all systems in the domain

• DNSSEC benefits:
  • Spoofing, cache poisoning immediately detectable
  • Minimizes overhead of doing so
    • No associated PKI defined
    • No key revocation mechanism defined (but can just change DNS server’s public, private keys)
Cookies

• Token containing information about state of transaction on network
  • Usual use: refers to state of interaction between web browser, client
  • Idea is to minimize storage requirements of servers, and put information on clients
• Client sends cookies to server
Some Fields in Cookies

- **name, value**: name has given value
- **expires**: how long cookie valid
  - Expired cookies discarded, not sent to server
  - If omitted, cookie deleted at end of session
- **domain**: domain for which cookie intended
  - Consists of last $n$ fields of domain name of server
  - Must have at least one “.” in it
- **secure**: send only over secured (TLS, HTTPS) connection
Example

• Caroline puts 2 books in shopping cart at books.com
  • Cookie: name bought, value BK=234&BK=8753, domain .books.com

• Caroline looks at other books, but decides to buy only those
  • She goes to the purchase page to order them

• Server requests cookie, gets above
  • From cookie, determines books in shopping cart
Who Can Get the Cookies?

• Web browser can send *any* cookie to a web server
  • Even if the cookie’s domain does not match that of the web server
  • Usually controlled by browser settings

• Web server can *only* request cookies for its domain
  • Cookies need not have been sent by that browser
Where Did the Visitor Go?

- Server books.com sends Caroline 2 cookies
  - First described earlier
  - Second has name “id”, value “books.com”, domain “adv.com”
- Advertisements at books.com include some from site adv.com
  - When drawing page, Caroline’s browser requests content for ads from server “adv.com”
  - Server requests cookies from Caroline’s browser
  - By looking at value, server can tell Caroline visited “books.com”
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

- Enciphered with public key for remailer #1
  - Remailer #2 address
  - Packet ID: 135
  - Symmetric key: 1

- Enciphered with symmetric encryption key #1

- Enciphered with public key for remailer #2
  - Final hop address
  - Packet ID: 168
  - Message ID: 7839
  - Symmetric key: 2
  - Random garbage

- Enciphered with symmetric encryption key #2
  - Recipient’s address
  - Any mail headers to add
  - Message
  - Padding if needed
Onion Routine

• 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

\{ \text{expires} \mid \mid \text{nexthop} \mid \mid E_F \mid \mid k_F \mid \mid E_B \mid \mid k_B \mid \mid \text{payload} \} \text{ pub}_r

- \text{payload}: data associated with message
- \text{expires}: expiration time for which \text{payload} is to be saved
- \text{nexthop}: node to forward message to
- \text{pub}_r: public key of next hop (node)
- \text{E}_F, k_F: encryption algorithm, key to be used when sending message forward to server
- \text{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