6.033 Computer System Engineering
Spring 2009

Nicolai Zeldovich

high-level ideas for today:
    attacks on network protocols
    principles for building secure protocols
    secure comm. channel abstraction; encryption and MAC
    authorization: lists vs tickets
    how to put it all together (security in HTTP)

Protocol Security

SLIDE: reminder of Denning-Sacco protocol

--- board 1 ---

protocol-level attacks
    impersonation: passwords are particularly bad [if used on multiple sites]
    replay
    reflection

SLIDE: reminder of how Denning-Sacco is broken
SLIDE: how to fix Denning-Sacco

protocol goals [right side of the board]
    appropriateness [explicitness?]
      explicit context, name principals
    forward secrecy [compromised keys do not reveal past communication]
      session keys, version numbers
    freshness [distinguish old and new messages]
      [later:] timestamp, nonce

--- board 2 ---

replay attack example:
    C -> S: "Buy 100 shares of GOOG", HMAC(m, pw)
    attacker retransmits the message many times!

    what's the fix? include a nonce, sequence number, timestamp, ..

--- board 3 ---

reflection attack example
    suppose Alice and Bob share a secret (e.g. password)
    want to ensure that the other party is present / alive
    shared key K between A & B

    A -> B: ENCRYPT(Na, K)
    B -> A: Na

    B -> A: ENCRYPT(Nb, K)
    A -> B: Nb

    attack: Lucifer pretends to be Alice, whereas Alice is really disconnected
      Bob asks Lucifer to decrypt challenge using their shared key
      Lucifer doesn't have the key, but can ask Bob to prove himself to Alice

Bob will decrypt his own challenge and send it to Lucifer!

--- board 4 ---

hard to reason about all these individual things when building a system
these building blocks are too low-level
common solution: abstraction of a secure communication channel

C <===> S

provides either confidentiality & authentication, or just authentication

+ long-lived comm
- short-lived sessions, many nodes
- offline messages, outside the model [e.g. confidentiality-only]

for -: have to use the lower-level abstractions we've been talking about

plan: use some protocol like denning-sacco to establish session key
or two keys, one for encryption, one for MACs
use session key for symmetric crypto for secure comm channel

need to include sequence numbers in messages to avoid replay, splicing

example: SSL/TLS in browsers
SLIDE: overview of SSL/TLS from wikipedia

[[encryption attacks:
ciphertext-only
known-plaintext
chosen-plaintext
chosen-ciphertext
]]

SLIDE: just for fun, watch what's happening on the wireless network
explain what tcpdump does

SLIDE: explain what we get out of it: packet headers, etc

SLIDE: gmail not encrypted

SLIDE: facebook not encrypted; plaintext

why don't people use SSL everywhere?
they probably should
some technical reasons: performance
bandwidth, CPU to encrypt messages is negligible now
but SSL requires handshake: 2 more RTTs

--- board 5 ---

secure comm channel =/= security
lots of trust assumptions underlying security of comm channel
claim: you're sending/receiving data to/from a particular entity
who is the entity? someone that a CA verified as being that entity
show amazon.com's certificate

```
      who are all of the CAs?  show list of all CAs in a browser?
          Edit -> Prefs -> Security -> List Certificates
      where do these CAs come from?  comes with browser from Mozilla or MS
    what can go wrong?
      maybe my trust in the name is misplaced (confusing name?  amaz0n.com?)
      maybe CA failed to properly verify the name?
      maybe one of the CAs was compromised?
      maybe one of the CA's keys was disclosed, stolen, guessed?
      your laptop was compromised, someone added an extra trusted CA root?
          alternatively, just modified your browser to do arbitrary things
    higher-level problems
      credentials leaked
      incorrect access control

--- board 6 ---

authorization
    lifetime of authorization:
      1. authorization
      2. mediation
      3. revocation

    revocation --\
    authorize -\ |
               V V
    client -> guard -> server -> object
    guard checks whether request should be allowed

--- board 7 ---

how does this guard work?
    logically, an access matrix

        F1 F2 F3
    A   R  R  RW
    B   RW RW --
    C   -- RW R

    principals on one axis -- need authentication
    objects on another axis -- guard needs to understand what's going on
    entries are allowed operations

where does this matrix get stored?  [right side of board]
    lists (ACLs in Unix, AFS): store column with the object
        authentication: figure out who the user is
      authorization: add user to ACL
        mediation: look up that user's entry in the object's column
    tickets (Java ptr, some URL, some cookies, Kerberos, certs): user stores
row
        must ensure the user can't tamper with his own row!
        authorization: out of scope -- however the user got the ticket (eg.
pw)
        mediation: look up current object in the user's supplied row

--- board 8 ---
```

```
    advantage: ticket model allows delegating privileges easily
    advantage: decouples authorization check from authentication check
        can change the two parts of the system separately
    disadvantage: need to get ticket first
    disadvantage: revocation is difficult
        plan A: chase down every ticket with each user
        plan B: invalidate all outstanding tickets, require users that are
            still authorized to get new tickets

--- board 9 ---

good ways to generate ticket?
    HMAC is easy
    [ resource/rights, gen#/timeout?, ... ], HMAC'ed with server key
    gen# supports revocation
    timeout similarly helps control ticket distribution [Kerberos]

    so let's see how this would work in practice
      ticket to access directory /mit/6.01
      expires 11/11/2009

      ticket = { /mit/6.01, 11/11/2009, MAC(/mit/6.01 + 11/11/2009) }
      ticket = { /mit/6.011, 1/11/2009, MAC(/mit/6.011 + 1/11/2009) }

      CRUCIAL: ensure all signed messages are unambiguously marshalled!

capabilities = naming + tickets
    combine names and tickets together
    can't talk about an object without also talking about the rights to it
    would have avoided the symlink attack

    file descriptors in Unix are sort-of like this

managing this matrix can be hard (either rows or columns)
    common simplification: roles or groups

--- board 10 ---

how are all these components put together?
    example: web browser/server interaction
    secure comm channel: SSL
    client (browser) authenticates the server's name
    server authenticates client's certificate (MIT personal certificates)
    otherwise server gets a connection to someone that hasn't authenticated
yet

    message sequence
      server: identify yourself           \
      client: username, password           > SSL (usually)
      server: here's a cookie (ticket)    /
      client: request 1 and cookie (ticket)
      client: request 2 and cookie (ticket)
      ...

    SLIDE: plaintext cookies: enough to log into google calendar
    SLIDE: plaintext cookies 2: list of other sites
```

often these cookies are valid even after you change your password!
because it's just a signed ticket stating your username
good idea: include pw gen# in cookie/ticket

tickets and ACLs not as different as they sound
tickets often issued based on underlying ACLs
ACLs often used to decide higher-level ops based on lower-level tickets
e.g. server certificate = ticket for server's principal
but other checks apply to server's name
client cert/cookie = ticket for that user's client privileges
but other checks apply to client's name