6.033 Computer System Engineering
Spring 2009

```
Security intro
Nickolai Zeldovich
==============

key ideas for today:
    key to security is understanding what the attacker can do
    principles: reduce trust (least privilege), economy of mechanism

--- board 1 ---

security / protection
    permeates computer system design
    if you don't design it right upfront, can be hard to fix later
    much like naming, network protocols, atomicity, etc
    will affect all of the above

give examples of security problems
    SLIDE: general security stats
      critical problems that allow attackers to take control over
      windows machines -- about once a week
    SLIDE: not surprising, then, that china controls computers in embassies
    SLIDE: even medical devices like pacemakers are vulnerable to attacks

so what is protection?  back to first board
    prevent access by bad guys
    allow access by good guys
    policies [lots of them, and we can't really cover all possibilities]
    [.. so much like with other topics] -> mechanism

--- board 2 ---

real world vs computer security
    same:
      lock - encryption
      checkpoints - access control
      laws, punishment
    different:
      attacks are fast, cheap, scalable
          ~same effort to compromise 1 or 1,000,000 machines
          can compromise machines all over the world
          no need to physically be present at the machine
          no strong notion of identity
      global; few laws

--- board 3 ---

policy goals: positive vs negative
    Nickolai can read grades.txt -- easy
      why easy?  build system, if it doesn't work, keep fixing until it does
    John cannot read grades.txt -- hard
      seems just the opposite of the above
      we can try asking John to log in and try to access grades.txt
      not enough: have to quantify all possible ways John might get grades.txt
          tries to access the disk storing the file directly
          tries to access it via a browser (maybe web server has access?)
          tries to read uninitialized memory after Nickolai's editor exits
```

tries to intercept NFS packets that are reading/writing grades.txt

tries to sell you a malicious copy of Windows
tries to take the physical disk out of server and copy it
tries to steal a copy of printout from the trash
calls the system administrator and pretends to be Nickolai
hard to say "regardless of what happens, John will not get grades.txt"

not enough to control access via one interface
must ensure all possible access paths are secure

we've seen some positive goals (e.g. naming) in 6.033 already
some negative goals too (transaction must not be "corrupted")
security is harder because attacker can do many things
with transactions, we knew what's going to happen (crash at any point)
most security problems are such negative goals

--- board 4 ---

threat model
the most important thing is to understand what your attacker can do
then you can design your system to defend against these things

C -> I -> S

typical setting:
client named Alice, server named Bob
an attacker (router) in the network, Eve, is eavesdropping
alternatively, Lucifer, a malicious adversary, can send, modify packets

does attacker control the client? server?

frequent assumption:
no physical, social engineering attacks
only intercept/send messages
might or might not compromise server, client

this picture applies even on a single machine
processes from diff. users making calls into the OS kernel

consider costs as well (both security and insecurity have a price)
convenience, HW expense, design, ..

right side of the board:
basic goals
- authentication [SLIDE: kentucky fax]
- authorization [who is authorized to release prisoners?]
- confidentiality NOTE: quite diff. from authentication!
- auditing
- availability

--- board 5 ---

policies / mechanisms
hardware: confine user code
mechanism: virtual memory, supervisor bit

```
      authentication: kernel initializes page table, supervisor bit
                  HW knows current state
      authorization: can access any virtual memory address in current PT
                   cannot access privileged CPU state
    Unix: private files
      mechanism: processes, user IDs, file permissions
      authentication: user password, when user starts a process
      authorization: kernel checks file permissions
    firewalls: restrict connections
      mechanism: packet filtering
      authentication: connection establishment
      authorization: list of allowed/prohibited connections
      seemingly weak mechanism, but surprisingly powerful in practice
    bank ATM: can only withdraw from your acct, up to balance
      mechanism: app-level checks in server process
      authentication: card & PIN
      authorization: track account balance
    cryptography: next lectures

--- board 6 ---

challenges
    bugs
      hard to build bug-free systems, write perfect code
      expect bugs, try to design your system to be secure despite them
      in recitation tomorrow, will look at some of these bugs
    complete mediation
      requires careful design
      SLIDE: paymaxx bug
    many mechanisms: hard to enforce coherent policy
      want to ensure that bank policies are followed
      what mechanisms do we have?
          virtual memory isolates processes
          kernel, file system implements ACLs
          bank ATM implements its own checks
          web banking might implement other checks
          system used by bank employees has other checks
          firewalls at different places in the network
    interactions between layers
      [caching/timing, naming, memory reuse, network replay]
      SLIDE: naming problem with symlink attacks
      SLIDE: password checking one character at a time

--- board 7 ---

safety net approach
    be paranoid -- make assumptions explicit
      attackers will try all possible corner cases
    consider the environment
      if you are relying on network security, check for open wireless networks
      if you are reusing, relying on another component, make sure it's secure
          code meant to run on non-networked system used on the web?
          never expected to deal with malicious inputs
    consider dynamics of use
      suppose only Nickolai should access grades.txt
          who can specify permissions for the grades file?
```

```
        who can modify editor on Athena?  or set permissions on it?
        who can control name translation for that file?
    defend in depth
      even if you have a server on a "secure" company network, still want
      to require passwords.  what if someone brings an infected laptop?

right side of the board:
    humans: weakest link
      - UI
      - safe defaults

--- board 8 ---

design principles
    open design, minimize secrets
      figure out what's going to differentiate bad guys vs good guys
      focus on protecting that, make everything else public
      authentication: ID public, sth. that proves you're that ID is secret
      SSNs, credit card numbers fail at this
          SSNs used both as ID and as credentials for authentication
          unclear what part of credit card number is really secret
          some receipts star-out first 12 digits, other star out last 4
    economy of mechanism
      simple security mechanism
      multiple security mechanisms interfere
      try hard to reduce security policies to existing mechanisms

      design to minimize "impedance mismatch" between security mechanisms
          usually a number of app layers between client and real object
          right side: diagram: Client-WebApp-FS-Disk
          suppose this is paymaxx which stores user tax data
          would be great if policy were enforced on obj directly
            then wouldn't have to trust the server app code
          suppose Obj is file -- mechanism is file permissions
            if diff users store their data in 1 file, can't use OS prot
            if we carefully design files 1 per user, may be able to use OS

    least privilege: minimize TCB
      TCB (trusted computing base)
      usually don't want to trust the network (next lectures will show how)
      break up your app into small components, each with least needed
privilege
```