Nathan Tung

CS111 Chapter 11 Overview, Section 11.1 Outline #10

Chapter 11 Overview and 11.1 (11-4 to 11-28)

- Security Overview
  - Security is a negative goal
  - Principles to follow: build safety net, be explicit (state assumptions), design for iteration (assume errors)
  - Questions to ask for trustworthiness
    - Authenticity? Integrity? Authorization?
  - Adversaries intentionally break into the system (as opposed to accidental damage)
  - Privacy: ability of an individual to determine if/when/to whom personal info is to be given
  - Security: techniques that protect information from unauthorized access (protection=security)
  - Separate mechanism from policy: principle that emphasizes mechanical rather than just social policy in designing a secure system
- Threat Classification
  - Three basic types
    - Unauthorized information release
      - Unauthorized person can read/use info stored in computer
    - Unauthorized information modification
      - Unauthorized person can make dangers changes in info or messages across a network
    - Unauthorized denial of use
      - Preventing authorized user from using info (system crash, denial of service, etc.)
  - Insider thread: legitimate user that acts kind of like an adversary
  - Examples: locking the computer, encouraging good security practices, encrypting credit card info, keeping logs for un-tampered audit trails (which help in court), etc.
  - “Authorized user can read file” vs. “Adversary cannot read file” demonstrates the negative problem
    - Designer has to come up with all possible threads (negative problem)
    - Designer needs to block all bad paths; adversary only needs to find one
- Safety Net
  - Be explicit
    - Write down all assumptions; don’t make assumptions that are highly dependent
  - Design for iteration
    - Assume we’ll make errors; be ready to iterate/implement the design/process
  - Other requirements
    - Certify system security (make sure design matches intended security policy and so on)
      - Usually done by independent reviewers
    - Maintain audit trails of authorization decisions
      - Feedback on incorrect assumptions/designs/implementations
    - Design the system for feedback
      - Environment that encourages feedback; reviewing logs and alert programs
  - Dynamics of use: establishment and change in who has access to what (or can change access)
  - Defend in depth: use redundant defenses; never believe your system is secure
- Design Principles
  - Open design principle: make the design public so everyone can comment/help on it
    - Minimize secrets; they won’t be secret for much longer
  - Complete mediation: force every access to require authentication and authorization
  - Principle of least astonishment: mediation/authorization should be transparent for user to understand
  - Adopt sweeping simplifications
  - Economy of mechanism: reduce the number of mechanisms so that the rest are correctly implemented
  - Minimize common mechanism: shared mechanisms provide unwanted communication paths
  - End-to-end argument: use the users’ level of certification; users can decide whether to use a substitute
  - Fail-safe defaults: access should be based on permission, not exclusion; lack of access should be default
  - Least privilege principle: use as little privileges as possible in a system or in running code
- High rate of change in technology poses security challenges
  - A lot of Internet and PC software don’t follow these principles

    - The PC was meant to be standalone with no security; designers didn't foresee the need for security on the computer network we have today
        - Hard to add security on after-the-fact
- Security Model
    - Complete mediation: system must mediate every action (including system ones) to be secure
    - Principal: entity inside the computer system responsible and accountable for being unit of authorization
        - Mediating an action asks: is the principal who requested the action authorized to perform it?
    - Two parts in a system
        - A guard (reference monitor) decides whether or not to do the action
        - Second part completes the action
    - Authentication vs Authorization
        - Verifying identity of principal vs. checking what objects are shared with whom
    - Cryptography creates a secure channel that protects communications over network
    - Even given "complete mediation," the adversary can circumvent guard, launch insider attack, overload the system with requests, etc.
        - Circumventing the guard
            - Find some other opening in the system (example :dial-up model line); use buffer overrun attack
        - Insider attacks
            - Guess principal password; bribe principal to act on his behalf; trick him to run program
    - Preventative measures
        - Run all requested operations with least privileges
        - Maintain audit trail for every operation
        - Make copies, archive data in secure places
        - Occasionally make manual reviews on what principals need what access/privileges
        - Not much can be done for denial-of-service attacks, but use audit trails to track adversary
- Trusted computing base (TCB)
    - Goal: minimize mechanisms that need to be correct in order to guarantee a secure system
        - The economy of mechanism principle; safety net approach (be explicit, design for iteration)
    - Untrusted and trusted modules
        - We want trusted modules to be completely working so new modules can use them
        - These form the TCB; goal is to find a distinctive split
    - Establishing what modules are in the TCB can be tough
        - A compromised module might give adversary full privileges or otherwise unauthorized access
    - Large TCB is not necessarily good
        - It can limit the changes ordinary users make (in modules) or limits rate of system evolution
    - Finally, minimize chance of errors (by minimizing size of TCB) and maximize rate of error discovery (use design process with feedback)
    - Building a TCB
        - Specify (explicitly) security requirements (such as secure communication over networks)
        - Design minimal TCB using tools like authentication logic
        - Implement TCB using tools like a language that checks for buffer overrun attacks
        - Run TCB while trying to break security
    - Hardest part is verifying steps are consistent, design meets specs and resists attacks, implementation matches design, and system running is actually the one implemented (to prevent Trojans)
    - Hire tiger team to find loopholes, although there is no guarantee that they'll find all the shortcomings
    - When a bug is found/repaired, the assumptions must be repaired as well
    - State what risks are acceptable (since it might not be worth implementing a negation to that risk)