THE UNIVERSITY of NEW ORLEANS

CSCI 4621/5621 Intro to CyberSecurity

00: Design Principles for Cyber Security

"*Trust, but verify*"

Vassil Roussev

vassil@cs.uno.edu

READING: Oorschot [ch1]

Design principles summary (per Oorschot)

- P1 simplicity-and-necessity
- P2 safe-defaults
- P3 open-design
- P4 complete-mediation
- P5 isolated-compartments
- P6 least-privilege
- P7 modular-design
- P8 small-trusted-bases
- P9 time-tested-tools
- P10 least-surprise
- P11 user-buy-in
- P12 sufficient-work-factor
- P13 defense-in-depth
- P14 evidence-production
- P15 datatype-validation
- P16 remnant-removal
- P17 trust-anchor-justification
- P18 independent-confirmation
- P19 request-response-integrity
- P20 reluctant-allocation

P1 SIMPLICITY-AND-NECESSITY

- Keep designs as simple and small as possible.
- Reduce the number of components used, retaining only those that are essential.
- Minimize functionality, favor minimal installs, and disable unused functionality.
- Economy and frugality in design—especially for core security mechanisms—simplifies analysis and reduces errors and oversights.
- Configure initial deployments to have non-essential services and applications disabled by default

P2 SAFE-DEFAULTS

- Use safe default settings
  - » *defaults often go unchanged*
- For access control, deny-by-default.
- Design services to be fail-safe
  - » *that is, when they fail, they fail "closed" (e.g., denying access).*
- Use explicit inclusion via allowlists (goodlists) of authorized entities with all others denied, rather
- than exclusion via denylists (badlists) that would allow all those unlisted

P3 OPEN-DESIGN

- Do not rely on secret designs, attacker ignorance, or security by obscurity.
- Invite and encourage open review and analysis.
  - *e.g., AES, SHA-3 were selected from a set of public candidates by open review*
- Leverage unpredictability where advantageous
  - *arbitrarily publicizing tactical defense details is rarely beneficial*
  - *beware exposing error messages or timing data that vary based on secret values*

P4 COMPLETE-MEDIATION

- For each access to every object verify proper authority.
  - *ideally **immediately** before the access is to be granted.*
- Verifying authorization requires
  - *authentication (corroboration of an identity),*
  - *checking that the associated principal is authorized, and*
  - *checking that the request has integrity*
    - it must not be modified after being issued by the legitimate party

P5 ISOLATED-COMPARTMENTS

- Compartmentalize system components using strong isolation structures (containers) that
  - *manage or prevent cross-component communication, information leakage, and control.*
- This limits damage when failures occur, and protects against escalation of privileges (Chapter 6).
- Restrict authorized cross-component communication to observable paths with defined interfaces to aid mediation, screening, and use of chokepoints.
- Examples of containment mechanisms:
  - *process and memory isolation,*
  - *disk partitions,*
  - *virtualization,*
  - *software guards,*
  - *zones, gateways and firewalls.*

P6 LEAST-PRIVILEGE

- Allocate the fewest privileges needed for a task, and for the shortest duration necessary.
- Example
  - *retain superuser privileges **only** for actions requiring them;*
  - *drop and reacquire privileges if needed later.*
  - *Do not use a Unix **root** account for tasks where regular user privileges suffice.*
- This reduces exposure, and limits damage from the unexpected
  - *such as software bugs*

P7 MODULAR-DESIGN

- Avoid monolithic designs that embed full privileges into large single components.
- Favor object-oriented and finer-grained designs that segregate privileges (including address spaces) across smaller units or processes.

P8 SMALL-TRUSTED-BASES

- Strive for small code size in components that must be trusted
  - *i.e., components on which a larger system strongly depends for security.*
- Example
  - *high-assurance systems centralize critical security services in a minimal core operating system microkernel*

P9 TIME-TESTED-TOOLS

- Rely wherever possible on time-tested, expert-built security tools including protocols, cryptographic primitives and toolkits,
  - » *rather than designing and implementing your own.*
- History shows that security design and implementation is difficult to get right even for experts;
  - » *thus amateurs are heavily discouraged (don't reinvent a weaker wheel).*
- Confidence increases with the length of time mechanisms and tools have survived under load.
  - » *not foolproof, e.g.,* ***Heartbleed*** (CVE-2014-0160)

P10 LEAST-SURPRISE

- Design mechanisms, and their user interfaces, to behave as users expect.
- Align designs with users' mental models of their protection goals, to reduce user mistakes that compromise security.
  - ***Especially*** *where errors are irreversible (e.g., sending confidential data or secrets to outside parties).*
- Tailor to the experience of target users
- Beware designs suited to experts but triggering mistakes by ordinary users.
- Simpler, easier-to-use (i.e., usable) mechanisms yield fewer surprises.

P11 USER-BUY-IN

- Design security mechanisms that users are ***motivated*** to use rather than bypass, and so that

  users' *path of least resistance is a safe path*.

- Seek design choices that
  - *illuminate benefits,*
  - *improve user experience, and*
  - *minimize inconvenience.*
- Mechanisms viewed as *time-consuming*, *inconvenient* or *without perceived benefit* risk non-compliance.

P12 SUFFICIENT-WORK-FACTOR

- For configurable security mechanisms
  - » *where the probability of attack success increases predictably with effort,*

  tune the mechanism so that
  - » *the cost to defeat it (work factor) clearly exceeds the resources of anticipated classes of adversaries.*
- That is, use ***suitably strong defenses***

P13 DEFENSE-IN-DEPTH

- Build defenses in multiple layers backing each other up,
  - *forcing attackers to defeat independent layers,*
  - *thereby avoiding single points of failure.*
- If any layer relies on alternative defense segments,
  - *design each to be comparably strong ("equal-height fences" for defense-in-breadth)*

  strengthen the weakest segment first
- As a design assumption,
  - *expect some defenses to fail on their own due to errors, and that*
  - *attackers will defeat others more easily than anticipated,*
  - *or entirely bypass them.*

P14 EVIDENCE-PRODUCTION

- Record system activities through event logs and other means to
  - *promote accountability,*
  - *help understand and recover from system failures, and support intrusion detection tools.*
- Example:
  - *robust audit trails support forensic analysis tools, to help reconstruct events related to intrusions and criminal activities.*
- In many cases, mechanisms that facilitate detection and evidence production may be more cost-effective than outright attack prevention.

P15 DATATYPE-VALIDATION

- Verify that all received data meets expected properties or data types
- If data input is expected, ensure that it

  cannot be processed as code by subsequent components.
- This may involve canonicalization of data and/or

  broader input sanitization

P16 REMNANT-REMOVAL

- On termination of a session or program,
  - » *remove all traces of sensitive data associated with a task,*
  - » *including secret keys and*
  - » *any remnants recoverable from*
    - secondary storage,
    - RAM and
    - cache memory.
- Note that common file deletion removes directory entries,
  - » *whereas secure deletion aims to make file content unrecoverable even by forensic tools.*
- Note that while a process is active, information may leak elsewhere by side channels.

P17 TRUST-ANCHOR-JUSTIFICATION

- Ensure/justify confidence placed in any base point of assumed trust,
  - *especially when mechanisms iteratively or transitively extend trust from a base point*
    - e.g., CA certificate chain
- More generally, verify trust assumptions where possible,
- with extra diligence at
  - *registration,*
  - *initialization,*
  - *software installation, and*
  - *starting points in the lifecycle of a*
    - software application,
    - security key or
    - credential.

P18 INDEPENDENT-CONFIRMATION

- Use simple, independent (e.g., local device) crosschecks
  - » *to increase confidence in code or data,*
  - » *especially if it may arrive from*
    - outside domains or
    - over untrusted channels.

P19 REQUEST-RESPONSE-INTEGRITY

- Verify that responses match requests
  - » *in name resolution and other distributed protocols.*
- Their design should include cryptographic integrity checks that

  bind steps to each other within a given transaction or protocol run

  to detect unrelated or substituted responses;
- Beware protocols lacking authentication.

P20 RELUCTANT-ALLOCATION

- Be reluctant to allocate resources or expend effort in interactions with unauthenticated, external agents.
- For services open to all parties, design to mitigate intentional resource consumption.
- Place a higher burden of effort on agents that initiate an interaction

Higher-level principles

HP1 SECURITY-BY-DESIGN

- Build security in, starting at the initial design stage of a development cycle
  - » *secure design often requires core architectural support absent if security is a late-stage add-on*
- Explicitly state the design goals of security mechanisms and what they are ***not*** designed to do
  - » *it is impossible to evaluate effectiveness without knowing goals*
- In design and analysis documents, explicitly state
  - » *all security-related assumptions*
    - especially related to trust and trusted parties
- Note that a security policy itself might not specify assumptions

HP2 DESIGN-FOR-EVOLUTION

- Design base architectures, mechanisms, and protocols to support evolution
  - *including algorithm agility for graceful upgrades of crypto algorithms*
  - *e.g., encryption, hashing*
- Support automated secure software update where possible
- Regularly re-evaluate the effectiveness of security mechanisms
  - *in light of evolving threats, technology, and architectures*
- Be ready to update designs and products as needed