CS527 Software Security

Basic Principles

Mathias Payer

Purdue University, Spring 2018

Software Security

Allow inteded use of software, prevent unintended use that may cause harm.

Security principles

- **Confidentiality:** an attacker cannot recover protected data
- **Integrity:** an attacker cannot modify protected data
- **Availability:** an attacker cannot stop/hinder computation

Accountability/non-repudiation may be used as fourth fundamental concept. It prevents denial of message transmission or receipt.

Security analysis

Given a software system, is it secure?

- . . . it depends
- What is the attack surface?
- What are the assets? (How profitable is an attack?)
- What are the goals? (What drives an attacker?)

Attacks and Defenses

- Attack (threat) models
  - A class of attacks that you want to stop
    - What is the attacker's capability?
    - What is impact of an attack?
    - What attacks are out-of-scope?
- Defenses address a certain attack/threat model.
  - General: e.g., stop memory corruptions
  - Very specific: e.g., stop overwriting a return address

Threat model

A threat model defines an attacker's abilities and resources.

- Awareness of entry points (and associated threats)
- Look at systems from an attacker's perspective
  - Decompose application: identify structure
  - Determine and rank threats
  - Determine counter measures and mitigation
- Reading material: `https://www.owasp.org/index.php/Application_Threat_Modeling`

Threat model: example

Assume you want to protecy your valuables by locking them in a safe.

- In trust land, you don't need to lock your safe.
- An attacker may pick your lock.
- An attacker may use a torch to open your safe.
- An attacker may use advanced technology (x-ray) to open the safe.
- An attacker may get access (or copy) your key.

Cost of security

There is no free lunch, security incurs overhead.
Security is...

- expensive to develop,
- may have performance overhead,
- may be inconvenient to users.

Fundamental security mechanisms

- Isolation
- Least privilege
- Fault compartments
- Trust and correctness

Figure 1

Isolation

Isolate two components from each other. One component cannot access data/code of the other component except through a well-defined API.

Least privilege

The principle of least privilege ensures that a component has the least privileges needed to function.

- Any privilege that is further removed from the component removes some functionality.
- Any additional privilege is not needed to run the component according to the specification.
- Note that this property constrains an attacker in the privileges that can be obtained.

Fault compartments

Separate individual components into smallest functional entity possible. General idea: contain faults to individual components. Allows abstraction and permission checks at boundaries.

Note that this property builds on least privilege and isolation. Both properties are most effective in combination: many small components that are running and interacting with least privileges.

Fault compartments

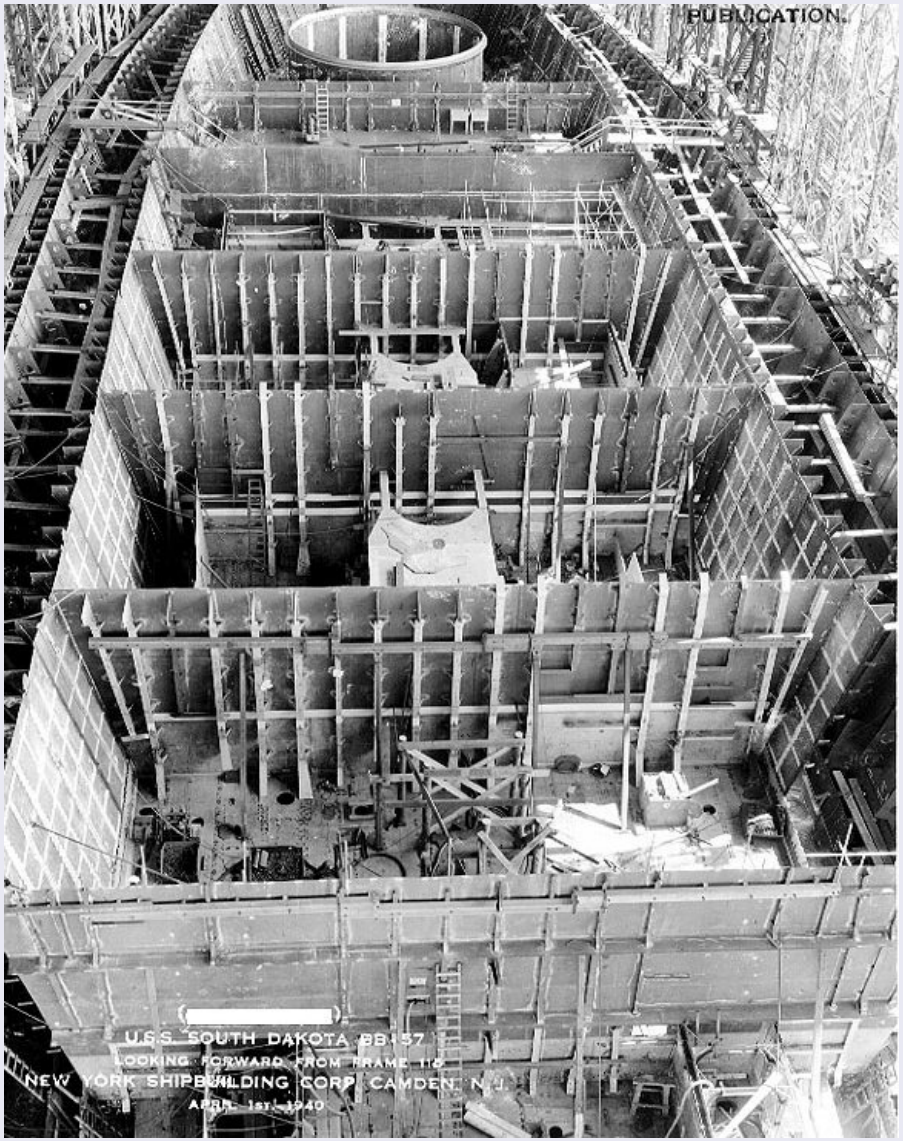

Figure 2

Trust and correctness

Specific components are assumed to be trusted or correct according to a specification.

Formal verification ensures that a component correctly implements a given specification and can therefore be trusted. Note that this property is an ideal property that cannot generally be achieved.

Hardware and software abstractions

- Operating System (OS) abstractions
- Hardware abstractions

Operating System (OS) abstraction

- Provides process abstraction
- Well-defined API to access hardware resources
- Enforces mutual exclusion to resources
- Enforces access permissions for resources
- Restrictions based on user/group/ACL
- Restricts attacker

Hardware abstraction

- Virtual memory through MMU/OS
- Only OS has access to raw physical memory
- DMA for trusted devices
- ISA enforces privilege abstraction (ring 0/3 on x86)
- *Hardware abstractions are fundamental for performance*

Access control

- **Authentication:** Who are you (something you know, have, or are)?
- **Authorization:** Who has access to object?
- **Audit/Provenance:** I'll check what you did.

Types of access control

- Discretionary Access Control (DAC): Object owners specify policy
- Mandatory Access Control (MAC): Rule and lattice-based policy
- Role-Based Access Control (RBAC): Policy defined in terms of roles (sets of permissions), individuals are assigned roles, roles are authorized for tasks.

DAC

- User has authority over resources she owns
- User determines permissions for her data if other users want to access it

MAC

- Centrally controlled
- One entity controls what permissions are given
- Users cannot change policy themselves

RBAC

- Access permission is broken into sets of roles.
- Users get assigned specific roles
- Administration privileges may be a role.

Different security models

- Access control lists (static)
- Capabilities (static)
- Bell-LaPadula (state machine)
- Information flow (flow dependent)

Access control matrix

Provide access rights for subjects to objects.

	foo	bar	baz
user	rwx	rw	rw
group	rx	r	r
other	rx		r

Used, e.g., for Unix/Linux filesystems or Android/iOS/Java security model for privileged APIs.
Introduced by Butler Lampson in 1971
`http://research.microsoft.com/en-us/um/people/blampson/08-Protection/Acrobat.pdf`.

Summary and conclusion

- Software security goal: allow intended use of software, prevent unintended use that may cause harm.
- Three principles: Confidentiality, Integrity, Availability.
- Security of a system depends on its threat model.
- Isolation, least privilege, fault compartments, and trust as concepts.
- Security relies on abstractions to reduce complexity.
- Reading assignment: Butler Lampson, Protection `http://doi.acm.org/10.1145/775265.775268`