Basic Principles

Yajin Zhou (http://yajin.org)

Zhejiang University

Credits: Most of slides are from cse443 of PSU (Trent Jaeger), CS-412 of EPFL(mathias payer), CMPSC 447 of PSU(Gang Tan)

What is security?

- "the property that a system behaves as expected"
  - G. Spafford and many others ....

- Note that this does not say what a system should or should not do.
  - Implication -- there is no universal definition or test for security (why?)
  - Apply this definition to the ATM
    - How do you think an ATM should behave?
    - What should it do?
    - What should it not do?

- We talk about expectations often in terms of *confidentiality*, *integrity*, and *availability*.

What is security?

- 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

Adversary

- An adversary is any entity trying to circumvent the security infrastructure
  - The curious and otherwise generally clueless (e.g., script-kiddies)
  - Casual attackers seeking to understand systems
  - Venal people with an ax to grind
  - Malicious groups of largely sophisticated users (e.g, spammers)
  - Competitors (industrial espionage)
  - Governments (seeking to monitor activities)

Trust

- Trust refers to the degree to which an entity is expected to behave
  - What the entity not expected to do?
    - E.g., not expose password
  - What the entity is expected to do (obligations)?
    - E.g., obtain permission, refresh

- A trust model describes, for a particular environment, who is trusted to do what?
- Note: you make trust decisions every day
  - Q: What are they?
  - Q: Whom do you trust?

Reflections on Trusting Trust

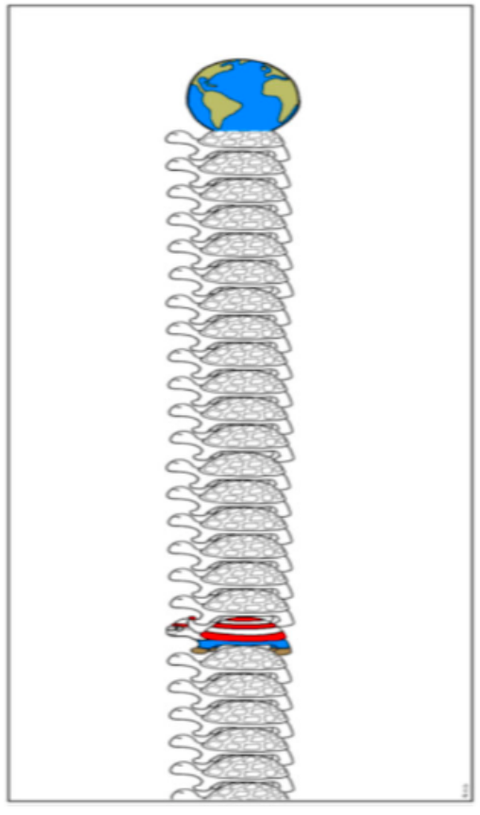
- Ken Thompson's Turing Award lecture describes the importance of the making clear what should be trusted
- Do you trust your compiler?
  - He describes an approach whereby he can generate a compiler that can insert a backdoor
    - e.g., insert a backdoor when recognizing a login program
- But you can examine the compiler source code
  - But, what program compiles the compiler?
  - He puts the malicious code in that program

Reflections on Trusting Trust

- Methodology
  - Generate a malicious binary that is used to compile compilers
  - Since the compiler source code looks **OK** and the malice is in the binary compiler compiler, it is difficult to detect.
- Such a program is an example of a Trojan horse malware

Turtles all the way down

- Take away: Thompson states the “obvious” moral that “you cannot trust code that you did not totally create yourself”
  - Creating a basis for trusting is very hard, **even today – why?**



A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever," said the old lady. "But it's turtles all the way down!“

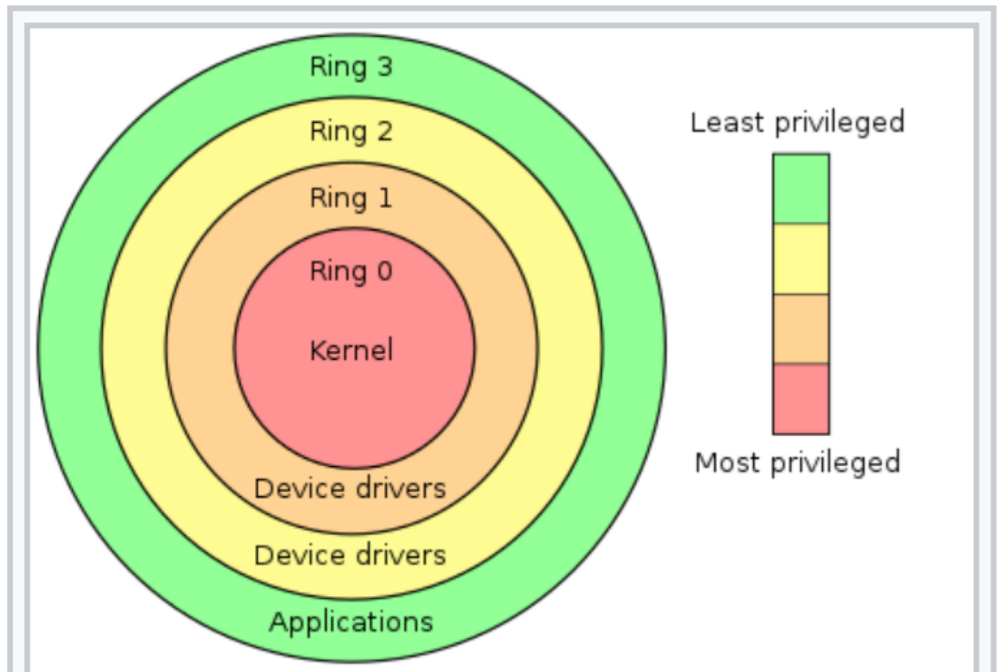
--- Stephen Hawking (1988), A Brief History of Time

Trusted Computing Base (TCB)

- A set of **hardware, firmware, and software that are critical to the security of a computer system**
  - Bugs in the TCB may jeopardize the system's security
  - E.g., a conventional e-voting machine: voting software + hardware
- **Components outside of the TCB can misbehave without affecting security**
- **In general, a system with a smaller TCB is more trustworthy**
- A lot of security research is about how to move components outside of the TCB (i.e., making the TCB smaller)
  - e.g., Proof-Carrying Code removes the compiler outside of the TCB
  - e.g., voter-verified paper ballots in E-voting

Trusted Computing Base (TCB)

- Fabrication of integrated circuits
- CPU designer
- Low level firmware
- Operating system
- Framework
- Application



Privilege rings for the x86. The ME is colloquially categorized as ring –3, below System Management Mode (ring –2) and the hypervisor (ring –1), all running at a higher privilege level than the kernel (ring 0)

Threats

- A threat is a specific means by which a risk can be realized by an adversary
  - Context specific (a fact of the environment)
  - An *attack vector* is a specific threat (e.g., access to install a keylogger)

- A threat model is a collection of threats that deemed important for a particular environment
  - E.g., should be addressed
  - A set of "*security requirements*" for a system

Threat model

- 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

> *The threat model defines the abilities and resources of the attacker. Threat models enable structured reasoning about the attack surface.*

Threat model example: OS

- Assume you want to protect 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 it.
  - An attacker may get access (or copy) your key.

Threat model example

Threat model: operating systems

- *Malicious extension:* inject an attacker-controlled driver into the OS;
- *Bootkit:* compromise the boot process (BIOS, boot sectors);
- *Memory corruption:* software bugs such as spatial and temporal memory safety errors or hardware bugs such as rowhammer;
- *Data leakage:* the OS accidentally returns confidential data (e.g., randomization secrets);
- *Concurrency bugs:* unsynchronized reads across privilege levels result in TOCTTOU (time of check to time of use) bugs;
- *Side channels:* indirect data leaks through shared resources such as hardware (e.g., caches), speculation (Spectre or Meltdown), or software (page deduplication);
- *Resource depletion and deadlocks:* stop legitimate computation by exhausting or blocking access to resources

Threat model example: malware

- Example: a piece of code as an attachment in an email
  - **What is trusted**: hardware + system software (e.g., OS)
  - **What is untrusted**: the attached code
    - Not sure what harm it will do
  - **Security objective**: protect the computer system against possibly malicious code
- Defenses
  - virus scanning; check for digital signatures; or just run it at your own risk

It’s not that obvious sometimes

- Protect a computer system from malicious code or malicious input
  - The system is trusted
- Protect software against malicious tampering
  - Code is trusted
  - But it is running in an untrusted system
- Bottom line
  - we need to carefully figure out the threat model: what to trust? ...

Policy and Enforcement

- Policy
  - What is (what is not) allowed
  - Who is allowed to do what
- **Enforcement**: what we do to cause policy to be followed
  - Means of enforcement
    - Persuasion, Monitoring & deterrence
    - Incentive management
    - **Technical prevention (what we are mostly interested in)**

Security Policies (CIA Model)

- Confidentiality: Info becomes known only to authorized people
  - Example: Bob buys 1,000 shares of Microsoft stocks and the info should be confidential
- Integrity: Info stored in a system is correct (not modified)
  - Example: Bill Gates sells 1M shares of MS stocks; the info is public. But nobody should change the number from 1M to 10M
- Availability: Info, or service, is available when you need it
  - Example: Bob wants to buy 1,000 shares of MS stocks from his broker, who happens to be unavailable (being ill)

Connecting them all

- Enforcement should aim to enforce a policy given threats in the threat model

Fundamental Security Mechanism

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

Isolation

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

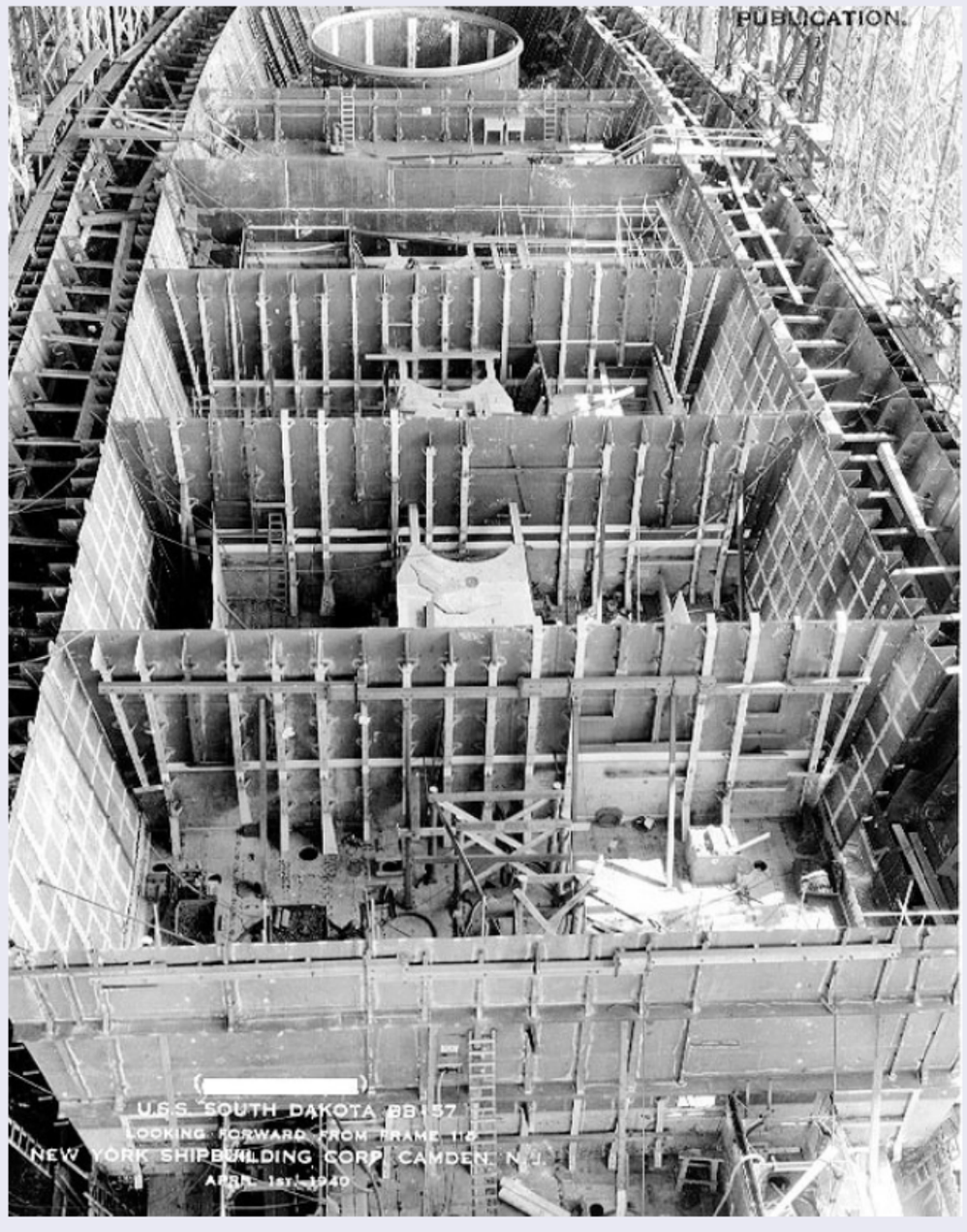
Least privilege

- 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.



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**.

- Problem: **how to ensure the correctness of the specification?**

- What if the specification is updated?

AAA

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

Authentication

- There are three fundamental types of identification:
  - What you know: username / password
  - What you are: biometrics
  - What you have: second factor / smartcard

Authorization

- An important question when handling shared resources is who can access what information.
  - These access policies are called access control models.
  - Access control models were originally developed by the US military
  - Users with different clearance levels on a single system
  - Data was shared across different levels of clearance
  - Therefore the name “multi-level security”

Authorization: Types of access control

- Mandatory Access Control (MAC): Rule and lattice-based policy
  - SELinux
  - MAC is controlled by administrators and requires lots of time and effort to maintain, but it provides a high level of security
- Discretionary Access Control (DAC): Object owners specify policy
  - Linux access control
  - DAC is much easier to implement and maintain, as users can manage access to the data they own.
- Role-Based Access Control (RBAC): Policy defined in terms of roles (sets of permissions), individuals are assigned roles, roles are authorized for tasks.

DAC example

Access control matrix

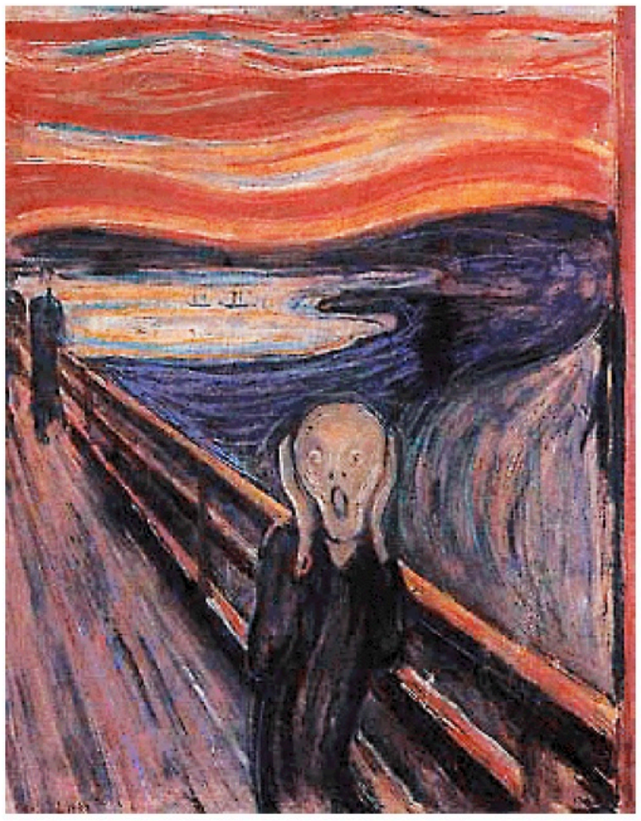
Provide access rights for subjects to objects.

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

- Used, e.g., for Unix/Linux file systems 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`.

Vulnerabilities

- A vulnerability is a flaw that is accessible (threat) to an adversary who has the capability to exploit that flaw
- Are flaws alone enough for a vulnerability?
  - E.g., buffer-overflow, WEP key reuse
- What is the source of a flaw?
  - Bad software (or hardware)
  - Bad design, requirements
  - Bad policy/configuration
  - System Misuse
    - unintended purpose or environment
    - E.g., student IDs for liquor store



Attacks

- An attack occurs when someone attempts to exploit a vulnerability
- Kinds of attacks
  - Passive (e.g., eavesdropping)
  - Active (e.g., password guessing)
  - Denial of Service (DOS)
    - Distributed DOS – using many endpoints

- A compromise occurs when an attack is successful
  - Typically associated with taking over/altering resources

Vulnerability Reporting

- Maintaining security is a difficult problem
  - Fortunately, people work together
  - .... by tracking vulnerabilities

    - Specify It: Open Vulnerability Assessment Language (OVAL)
    - Name It: CVE - Common Vulnerabilities and Exposures
    - Post It: Publicly accessible CVE List at
      - *cve.mitre.org*
    - Check for It: Readable and machine parseable (XML)
      - Can check for some of these automatically

- Common types of vulnerabilities become
  - CWE - Common Weakness Enumeration
    - Hierarchical organization
  - Publicly available
    - *cwe.mitre.org*

Why Security is Hard

Answer 2: If security is all about ensuring that *bad things never happen*, that means we have to know what those bad things are.

The hardest thing about security is convincing yourself that you've thought of all possible attack scenarios, before the attacker thinks of them.

"A good attack is one that the engineers never thought of."
–Bruce Schneier

If Security Gets in the Way

Security is meant to prevent bad things from happening; one side-effect is often to prevent useful things from happening.

Typically, a tradeoff is necessary between security and other important project goals: functionality, usability, efficiency, time-to-market, and simplicity.

Some Lessons

He who defends everything defends nothing. –old military adage

- Security is difficult for several reasons.
- Since you can never achieve perfect security, there is always a tradeoff between security and other system goals.

Summary

- Trust
- Threat model: define the resources and capabilities of attackers, who to trust
- Policy and enforcement: Enforcement should aim to enforce a policy given threats in the threat model
- Isolation, least privilege, fault compartments and trust
- Authentication, authorization and audit