CE/CS/SE 3354
Software Engineering

Software Security

What is computer security?

- Most developers and operators are concerned with **correctness**: achieving desired behavior
  - A working banking web site, word processor, blog, …
- Security is concerned with ***preventing*** **<u>un</u>desired behavior**
  - Considers an enemy/opponent/hacker/adversary who is *actively and maliciously* trying to *circumvent* any protective measures you put in place

Kinds of undesired behavior

- Stealing information: ~~**confidentiality**~~
  - Corporate secrets (product plans, source code, …)
  - Personal information (credit card numbers, SSNs, …)
- Modifying information or functionality: ~~**integrity**~~
  - Installing unwanted software (spyware, …)
  - Destroying records (accounts, logs, plans, …)
- Denying access: ~~**availability**~~
  - Unable to purchase products
  - Unable to access banking information

Significant security breaches

- **RSA**, March 2011
  - stole tokens that permitted subsequent compromise of customers using RSA SecureID devices
- **Adobe**, October 2013
  - stole source code, 130 million customer records (including passwords)
- **Target**, November 2013
  - stole around 40 million credit and debit cards
- … and many others!

Defects and Vulnerabilities

- Many breaches begin by exploiting a **vulnerability**
  - This is a *security-relevant* **software defect** that can be **exploited** to effect an undesired behavior
- A software **defect** is present when the software behaves incorrectly, i.e., it fails to meet its requirements
- Defects occur in the software's *design* and its *implementation*
  - A **flaw** is a defect in the design
  - A **bug** is a defect in the implementation

Example: RSA 2011 breach

- Exploited an Adobe Flash player vulnerability

1. A **carefully crafted Flash program**, when run by the vulnerable Flash player, allows the **attacker to execute arbitrary code** on the running machine
2. This program could be **embedded in an Excel spreadsheet**, and run automatically when the spreadsheet is opened
3. The spreadsheet could be attached to an **e-mail masquerading to be from a trusted party** (*spear phishing*)

Considering **Correctness**

- The Flash vulnerability is an implementation **bug**
  - All software is buggy. So what?
- A normal user never sees most bugs, or works around them
  - Most (post-deployment) bugs due to rare feature interactions or failure to handle edge cases
- Assessment: Would be too expensive to fix every bug before deploying
  - So companies only fix the ones most likely to affect normal users

Considering **Security**

Key difference:

An adversary is not a normal user!

- The **adversary will actively attempt to find defects** in rare feature interactions and edge cases
  - For a typical user, (accidentally) finding a bug will result in a crash, which he will now try to avoid
  - An adversary will work to find a bug and exploit it to achieve his goals

To ensure security, we must

***eliminate bugs** and **design flaws**, and/or*

*make them **harder to exploit***

What is Software Security?

Software Security

- Software security is a kind of computer security that focuses on the **secure design and implementation of software**
  - Using the best languages, tools, methods
- ***Focus*** *of study:*

`the code`

- By contrast: Many popular approaches to security treat software as a *black box* (ignoring the code)
  - OS security, anti-virus, firewalls, etc.

Why Software Security?

Firewalls and anti-virus are like building walls around a weak interior

Attackers often can bypass outer defenses to attack weaknesses within

Software Security aims to address weaknesses directly

Operating System Security

- Operating systems **mediate a program's actions**
  - *Aka* **system calls**
  - such as reading and writing files,
  - sending and receiving network packets,
  - starting new programs, etc.
- Enforceable policies control actions
  - programs run by Alice cannot read files owned by Bob
  - programs run by Bob cannot use TCP port 80
  - programs run in directory D cannot access files outside of D

Limitations of OS Security

- **Cannot enforce application-specific policies**, which can be too fine-grained
  - Example: database management system (DBMS)
- **Cannot** (precisely) **enforce info-flow policies**
  - An operating system typically implements an **execution monitor**: decisions are based on *past and current actions*
  - **Information flow policies**: A *non*-action may reveal something about a secret without leaking it directly

Anti-virus Scanners

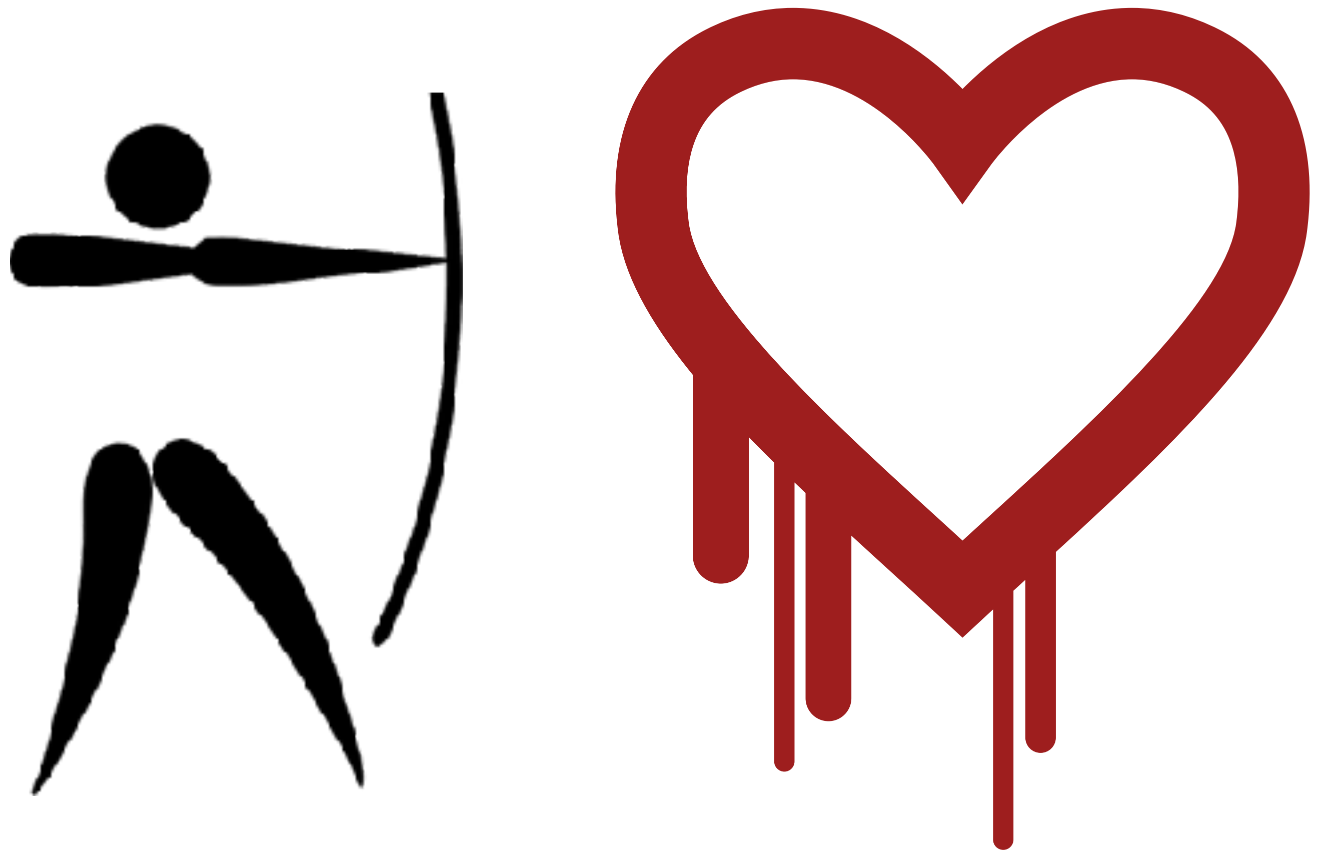
- Anti-virus scanners look for **signs of malicious behavior** in local **files**
- anti-virus looks for patterns
- Newer forms of anti-virus scanners are sophisticated, but **in practice are frequently bypassed**
  - **Trade off precision and performance** (latter could compromise availability)

Ex: Heartbleed

- SSL/TLS is a core **protocol** for **encrypted communications** used by the web
- Heartbleed is a **bug** in the commonly used **OpenSSL** implementation of SSL/TLS, v1.0.1 - 1.0.1f
  - Discovered in March 2014, it has been in released code since March 2012 (**2 years old!**)
- A carefully crafted packet causes OpenSSL to read and return portions of a vulnerable server's memory
  - Leaking passwords, keys, and other private information

Heartbleed, meet SoftSec

- **Black box security is incomplete against Heartbleed exploits**
  - Issue is not at the level of system calls or deposited files: nothing the OS or antivirus can do
- **Software security** methods attack the **source** of the problem: **the buggy code**



Secure Software Development

- Consider security *throughout software lifecycle*
  - **Requirements**
  - **Design**
  - **Implementation**
  - **Testing**

- Corresponding activities
  - **Define** *security requirements, abuse cases,*
  - **Perform** *architectural risk analysis (threat modeling)* and *security-conscious design*
  - **Conduct** *code reviews, risk-based security testing,* and *penetration testing*

Designing and Building **Secure Software**

Making secure software

- **Flawed approach**: Design and build software, and ***ignore security at first***
    - Add security once the functional requirements are satisfied

- **Better approach**: ***Build security in*** from the start
    - Incorporate security-minded thinking into all phases of the development process

Development process

- Many development processes; **four common phases**:
  - **Requirements**
  - **Design**
  - **Implementation**
  - **Testing**

- Where does **security engineering** fit in?
  - **All phases!**

Security engineering

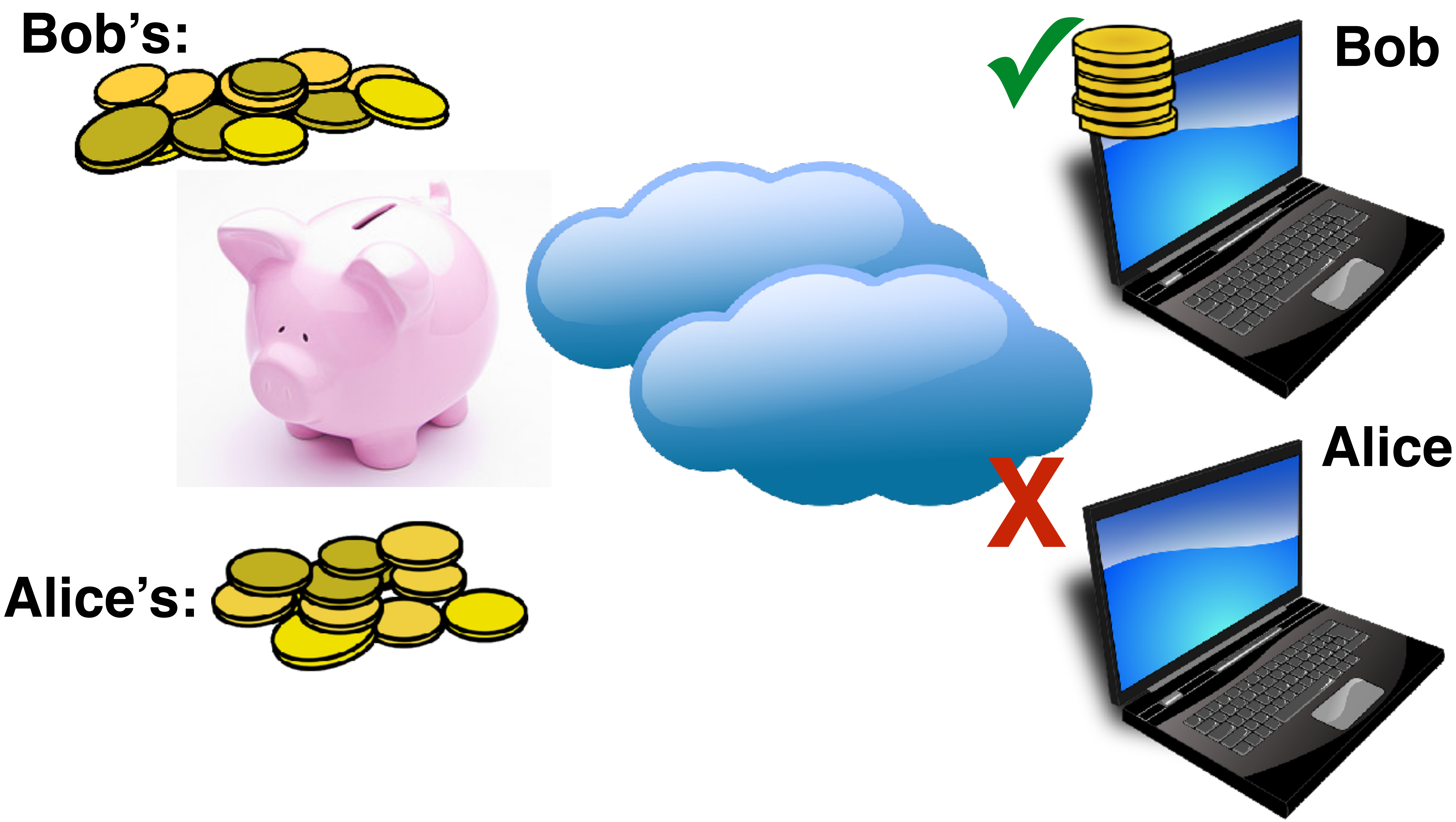
Phases

- **Requirements**
- **Design**
- **Implementation**
- **Testing**

Security Requirements
Abuse Cases
Architectural Risk Analysis
Security-oriented Design
Code Review (with tools)
Penetration Testing

Activities

Running Example: **On-line banking**



Threat Modeling
(Architectural Risk Analysis)

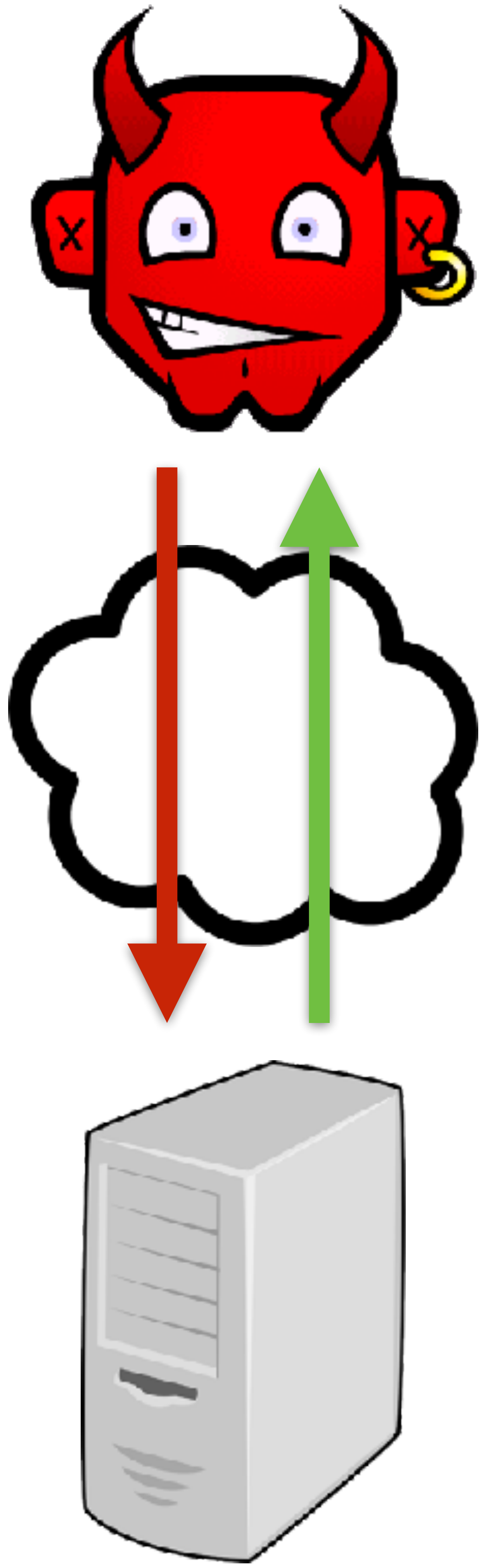
Threat Model

- The **threat model makes explicit the adversary's assumed powers**
  - Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong
- The threat model is **critically important**
  - If you are not explicit about what the attacker can do, how can you assess whether your design will repel that attacker?
- This is part of **architectural risk analysis**

Example: **Network User**

- An (anonymous) user that can connect to a service via the network
- Can:
  - **measure** the size and timing of requests and responses
  - run **parallel sessions**
  - provide **malformed inputs**, **malformed messages**
  - **drop or send extra messages**

- **Example attacks**: SQL injection, XSS, buffer overrun, …

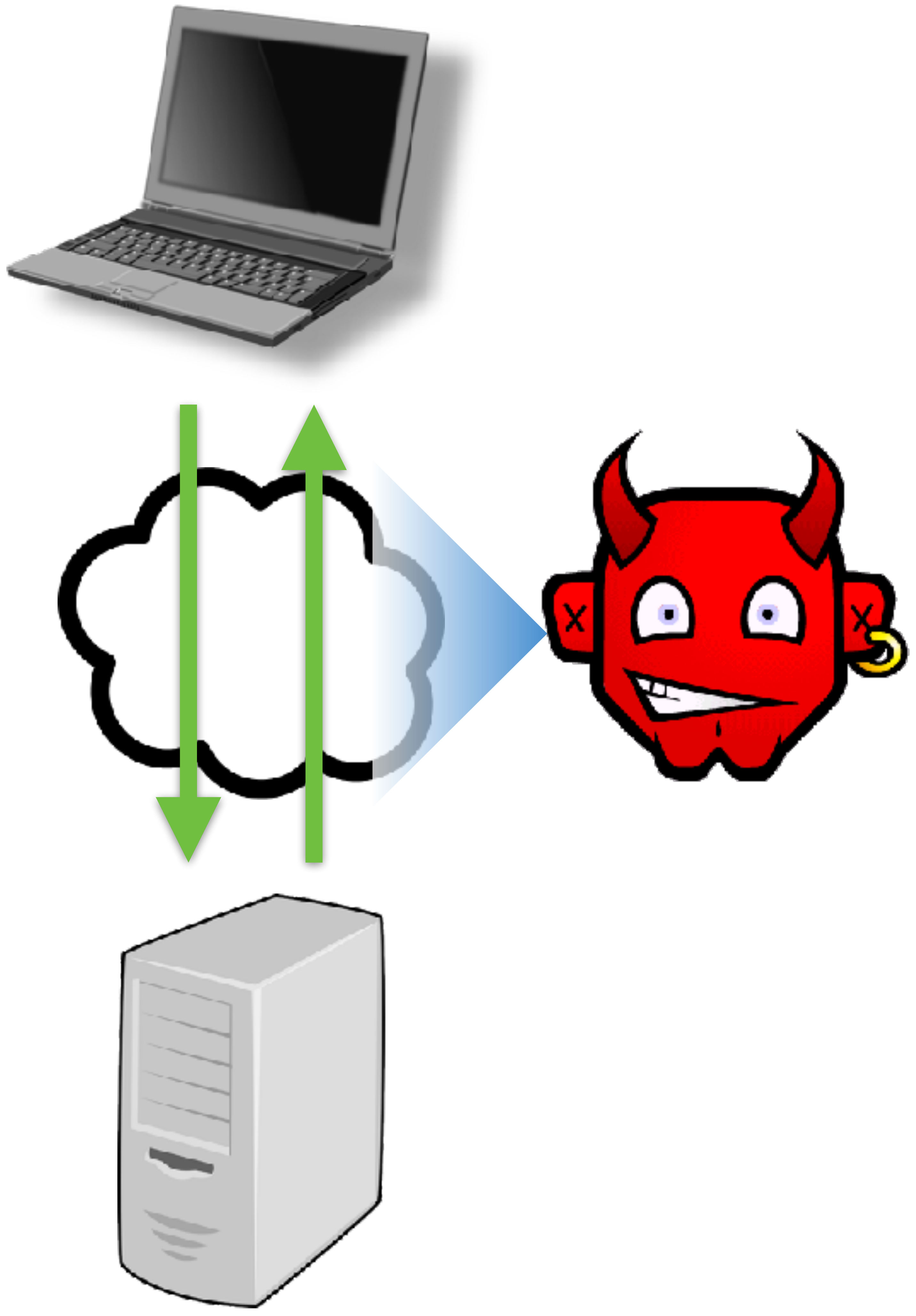


Example: **Snooping User**

- Internet user **on the same network** as other users of some service
  - For example, someone connected to an unencrypted Wi-Fi network at a coffee shop

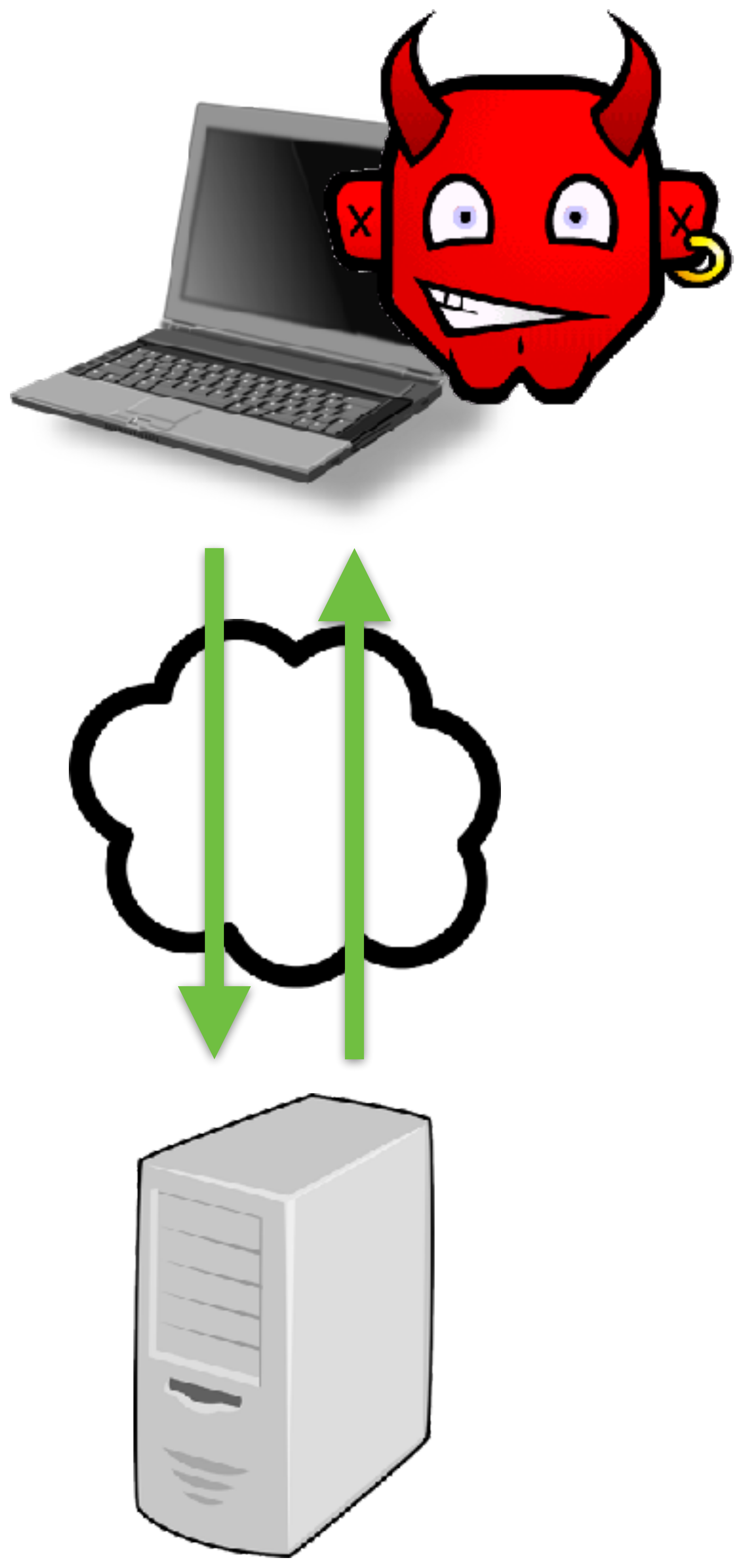
- Thus, can additionally
  - **Read/measure** others' **messages**,
  - **Intercept, duplicate,** and **modify messages**

- **Example attacks**: **Session hijacking** (and other data theft), **privacy-violating side-channel attack**, **denial of service**



Example: **Co-located User**

- Internet user **on the same machine** as other users of some service
  - E.g., **malware** installed on a user's laptop
- Thus, can additionally
  - **Read/write** user's **files** (e.g., cookies) and **memory**
  - **Snoop keypresses** and other events
  - Read/write the user's **display**
- **Example attacks: Password theft** (and other credentials/secrets)



Threat-driven Design

- Different threat models will elicit different responses

- **Network-only attackers** implies **message** traffic **is safe**
  - No need to encrypt communications

- **Snooping attackers** means **message** traffic **is visible**
  - So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)

- **Co-located attacker** can **access local files, memory**
  - Cannot store unencrypted secrets, like passwords

Bad Model = Bad Security

- Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

- E.g.: Assuming no snooping users **no longer valid**
  - *Prevalence of wi-fi networks in most deployments*
- Other mistaken assumptions
  - **Assumption**: Encrypted traffic carries no information
    - Not true! By analyzing the size and distribution of messages, you can infer application state
  - **Assumption**: Timing channels carry little information
    - Not true! Timing measurements of previous RSA implementations could be used eventually reveal a remote SSL secret key

Finding a good model

- **Compare against similar systems**
  - What attacks does their design contend with?
- **Understand past attacks and attack patterns**
  - How do they apply to your system?
- **Challenge assumptions in your design**
  - What happens if an assumption is untrue?
    - What would a breach potentially cost you?
  - How hard would it be to get rid of an assumption, allowing for a stronger adversary?
    - What would that development cost?

Security Requirements

Security Requirements

- **Software requirements** typically about **what** the **software should do**

- We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example**: One user's bank account balance should not be learned by, or modified by, another user, unless authorized
  - **Required mechanisms for enforcing them**
    - **Example**:
      1. Users identify themselves using passwords,
      2. Passwords must be "strong", and
      3. The password database is only accessible to login program.

Typical *Kinds* of Requirements

- **Policies**
  - **Confidentiality** (and Privacy)
  - **Integrity**
  - **Availability**

- Supporting **mechanisms**
  - **Authentication**
  - **Authorization**
  - **Auditability**

Privacy and Confidentiality

- *Definition*: **Sensitive information not leaked** to unauthorized parties
  - Called *privacy* for individuals, *confidentiality* for data

- **Example** policy: bank account status (including balance) known only to the account owner

- Leaking **directly** or via **side channels**
  - **Example**: manipulating the system to directly display Bob's bank balance to Alice
  - **Example**: determining Bob has an account at Bank A according to shorter delay on login failure

Secrecy vs. **Privacy**? https://www.youtube.com/watch?v=Nlf7YM71k5U

Integrity

- *Definition*: **Sensitive information not damaged** by (computations acting on behalf of) unauthorized parties

- **Example**: Only the account owner can authorize withdrawals from her account

- Violations of integrity can also be **direct** or **indirect**
  - **Example**: Being able specifically withdraw from the account vs. confusing the system into doing it

Availability

- *Definition*: A system is **responsive to requests**

- **Example**: a user may always access her account for balance queries or withdrawals

- **Denial of Service (DoS)** attacks attempt to **compromise availability**
  - by busying a system with useless work
  - or cutting off network access

Question

- An attacker defacing your personal web page is a violation of what policy?
- A. Confidentiality
- B. Privacy
- C. Integrity
- D. Availability

Question

- An attacker defacing your web page is a violation of what policy?
- A. Confidentiality
- B. Privacy
- **C. Integrity**
- D. Availability

Supporting mechanisms

- Leslie Lamport's **Gold Standard** defines mechanisms provided by a system to enforce its requirements
  - **Au**thentication
  - **Au**thorization
  - **Au**dit

Authentication

- What is the **subject of security policies**?
  - Need to define a ***notion of identity*** and a way to ***connect an action with an identity***
    - *a.k.a. a* **principal**

- **How can system tell a user is who he says he is?**
  - What (only) he **knows** (e.g., password)
  - What he **is** (e.g., biometric)
  - What he **has** (e.g., smartphone)
  - Authentication mechanisms that employ more than one of these factors are called **multi-factor authentication**
    - E.g., bank may employ passwords and text of a special code to a user's smart phone

Authorization

- Defines **when a principal may perform an action**
- **Example**: Bob is authorized to access his own account, but not Alice's account
- There are a wide variety of **policies** that define what actions might be authorized
  - E.g., access control policies

Audit

- Retain enough information to be able to **determine the circumstances of a breach or misbehavior** (or *establish one did not occur*)
    - Such information, often stored in **log files**, must be **protected from tampering**, and from access that might violate other policies

- **Example**: Every account-related action is logged locally and mirrored at a separate site

Question

- Video cameras at a bank enable what kind of security mechanism?
- A. Authentication
- B. Authorization
- C. Audit

Question

- Video cameras at a bank enable what kind of security mechanism?
- A. Authentication
- B. Authorization
- **C. Audit**

Defining Security Requirements

- Many processes for deciding security requirements

- Example: **General policy concerns**
  - Due to **regulations**/standards
  - Due **organizational values** (e.g., valuing privacy)

- Example: **Policy arising from threat modeling**
  - Which **attacks** cause the **greatest concern**?
    - Who are the likely adversaries and what are their goals and methods?
  - Which **attacks** have **already occurred**?
    - Within the organization, or elsewhere on related systems?

Abuse Cases

- Abuse cases illustrate security requirements
- Where use cases describe what a system *should* do, **abuse cases describe what it should *not* do**
- Example **use case**: The system shall allow bank managers to modify an account's interest rate
- Example **abuse case**: A user is able to spoof being a manager and thereby change the interest rate on an account

Defining Abuse Cases

- Using attack patterns and likely scenarios, construct cases in which an **adversary's exercise of power** could **violate a security requirement**
  - Based on the threat model
  - What might occur if a security measure was removed?

- **Example**: *Co-located attacker* steals password file and learns all user passwords
  - Possible if password file is not encrypted
- **Example**: *Snooping attacker* replays a captured message, effecting a bank withdrawal
  - Possible if messages are have no *nonce*

Design Flaws

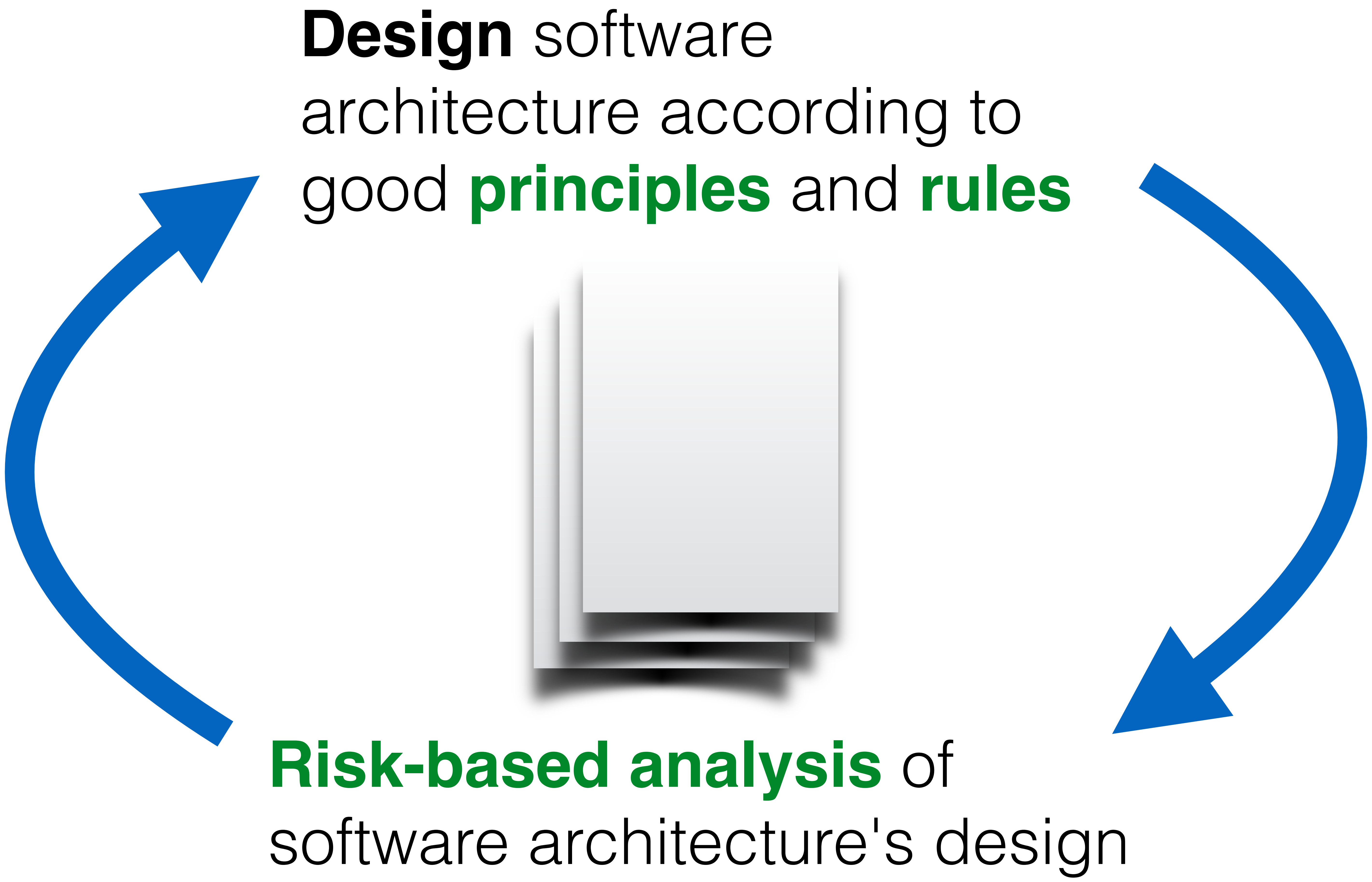
Design Defects = Flaws

- Recall that software defects consist of both flaws and bugs
  - **Flaws** are problems in the **design**
  - **Bugs** are problems in the **implementation**

- **We avoid flaws during the design phase**

- According to Gary McGraw,
  50% of security problems are flaws
  - So this phase is very important

Secure Software Design



Categories of Principles

- A **principle** is a high-level design goal with many possible manifestations
- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java
- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
  - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another
- **Detection** (and **Recovery**)
  - **Goal**: Identify and understand an attack (and undo damage)
  - **Example**: Monitoring (e.g., expected invariants), snapshotting

The Principles

- **Favor simplicity**
  - Use fail-safe defaults
  - Do not expect expert users
- **Trust with reluctance**
  - Employ a small trusted computing base
  - Grant the least privilege possible
    - Promote privacy
    - Compartmentalize
- **Defend in Depth**
  - Use community resources - no security by obscurity
- **Monitor and trace**

Design Category: Favor Simplicity

Favor Simplicity

- Keep it **so simple** it is **obviously correct**
  - Applies to the external interface, the internal design, and the implementation
  - **Category**: Prevention

"We've seen **security bugs in almost everything**: operating systems, applications programs, network hardware and software, and security products themselves. **This is a direct result of the complexity of these systems.** The more complex a system is--the more options it has, the more functionality it has, the more interfaces it has, the more interactions it has--the harder it is to analyze [its security]". — *Bruce Schneier*

https://www.schneier.com/essays/archives/1999/11/a_plea_for_simplicit.html

FS: Use fail-safe defaults

- Some **configuration** or **usage choices affect** a system's **security**
  - The length of cryptographic keys
  - The choice of a password
  - Which inputs are deemed valid

- **The default choice should be a secure one**
  - **Default key length is secure** (e.g., 2048-bit RSA keys)
  - **No default password**: cannot run the system without picking one
  - **Whitelist** valid objects, rather than blacklist invalid ones
    - E.g., don't render images from unknown sources

U.S.

The New York Times

Hackers Breach Security of HealthCare.gov

By ROBERT PEAR and NICOLE PERLROTH SEPT. 4, 2014

EMAIL
FACEBOOK
TWITTER
SAVE
MORE

WASHINGTON — Hackers breached security at the website of the government's health insurance marketplace, HealthCare.gov, but did not steal any personal information on consumers, Obama administration officials said Thursday.

. . .

Mr. Albright said the hacking was made possible by several security weaknesses. The test server should not have been connected to the Internet, he said, and it came from the manufacturer with a default password that had not been changed.

FS: Do not expect expert users

- Software designers should consider how the **mindset and abilities** of (the least sophisticated of) a system's **users** will **affect security**

- **Favor simple user interfaces**
  - **Natural or obvious choice is the secure choice**
    - Or avoid choices at all, if possible, when it comes to security
  - **Don't have users make frequent security decisions**
    - Want to avoid user fatigue

Passwords

- **Goal: easy to remember but hard to guess**
  - Turns out to be **wrong** in many cases!
    - **Hard to guess = Hard to remember!**

- **Password cracking tools** train on released data to quickly guess common passwords
  - John the Ripper, http://www.openwall.com/john/
  - many more …
  - **Top 10 worst passwords of 2013**:123456, password, 12345678, qwerty, abc123, 123456789, 111111, 1234567, iloveyou, adobe123

Password Manager

- A password manager (PM) stores a **database of passwords, indexed by site**
  - Encrypted by a **single, master password** chosen (and remembered) by the user, used as a key
  - **PM generates complicated per-site passwords**
    - Hard to guess, hard to remember, but the latter doesn't matter!

- **Benefits**
  - Only a single password for user to remember
  - User's password at any given site is hard to guess
  - Compromise of password at one site does not permit immediate compromise at other sites
- **But**:
  - Must still **protect** and **remember** strong **master password**

Password Strength Meter

- **Gives user feedback on the strength of the password**
  - Intended to **measure guessability**
  - Research shows that these **can work**, but the design must be **stringent** (e.g., forcing unusual characters)
    - Ur et al, “How does your password measure up? The effect of strength meters on password creation”, Proc. USENIX Security Symposium, 2012.