Lecture 26: Non-functional properties I

CS 5150, Spring 2022

Course reminders

- A4 due today
  - Any surprising vulnerabilities?
- Final presentations Fri-Tue
  - Focus on demonstration
    - Client must know if they should accept this code
  - Internal projects: work with your client to deploy your code
    - Most features should be merged by now
- Handoff package due Tue
  - Final report + user manual + maintenance guide + code
  - Don't forget peer evaluations

Lecture goals

- Identify common non-functional requirements areas
- Define building blocks of secure systems

Security

A one-lecture survey

Security goals

- Ensure availability for legitimate use
- Prevent malicious use
- Preserve privacy

- Security is a property of the entire system
  - Only as strong as the weakest link (which may not be software)
  - Very difficult to improve after system is designed
- Appropriate security measures requires threat modeling

The cost of poor security

- Annual cost of poor software quality: $2 trillion
  - 75% due to security vulnerabilities
- Annual cost of cybercrime: $6 trillion
  - Ransomware damages: $20 billion
- Escalating risk of cyber warfare

- One lecture will not make you an expert (see CS 5430)
  - Familiarize terminology and context
  - Avoid common mistakes

CISQ/Synopsys report (2020)
Cybersecurity Ventures (2020)

Perfect security is impractical

- At some point, risks are treated as a cost of doing business
  - Fines
  - Identity protection services
  - Loss of customers
  - Cost of required fixes
  - Loss of confidential information (trade secrets, export-controlled data)
  - Loss of system availability
  - Blackmail & ransom
  - Theft & fraud
- Example: credit card purchases
  - Users trust them because *banks limit their liability* in cases of fraud
  - Banks, merchants invest in technologies to *limit fraud to affordable levels*
    - Must balance security measures with consumer convenience, cost of deployment
    - Slow adoption of chip-based cards in USA

Economics of security

- Security is a difficult property to quantify/verify
- Secure development practices are expensive
  - Little evidence that customers will pay extra for better security
- Many risks are borne externally (by users)
- Overall system security limited by weakest link
- **But** software insecurities are scalable, brittle
  - Once discovered, exploitation may be rapid and widespread
  - Networked systems allow compromise to propagate to increasingly critical systems
- Future societal costs may be enormous

Minimizing risk

- System design
- Interface design
- Development practices
- Operating procedures
- Training

Supply chain vulnerabilities

- Software security depends on entire stack
  - Hardware
  - Operating system & services
  - Library dependencies
  - Build tools & runtime environment
  - Application itself
  - Trusted & secret data
- … for both client and server

- Risk management requires policy and planning
  - Stay up-to-date with security updates (may require downtime)
  - Stay up-to-date with revocation lists, certificate renewal
  - Balance availability against security (timeline for phasing out support for insecure versions)

Access control

Authentication (AuthN)

- Verifying a user's identity
  - Multi-factor
    - Knowledge (e.g. password)
    - Possession (e.g. SMS)
    - Biometric (e.g. fingerprint)
  - Need to pair identities with check data (e.g. password hash, public key)
    - Must establish trust
    - Want to minimize risk of spoofing

Authorization (AuthZ)

- Determine what a user is allowed to do
  - Filesystem permissions
  - Access Control Lists (ACLs)

Password-based authentication

- Storing passwords is a liability
  - Allows spoofing accounts with shared credentials
- Store password hashes instead
  - Incorporate a random salt to prevent certain attacks
  - Use a resource-intensive hash to slow down brute force attacks

- Password hashing standards: bcrypt, scrypt, PBKDF2, Argon2
- NIST guidelines advise against complexity and rotation requirements

Access control technologies

- OpenID/OIDC (authentication)
- OAuth 2.0 (delegate authorization)
  - Providers include major web service providers: Apple, Facebook, Google, Microsoft
- SSH keys (authentication)
- LDAP (identity, authentication, authorization)
- Kerberos (SSO authentication)
  - Supported by both Unix and Windows
- SAML/Shibboleth (web SSO authentication & authorization)

Network security & barriers

- In practice, access control is often linked to network presence
  - Denial-of-service attacks can affect resources without needing to authenticate
- Firewalls, private networks, and airgaps provide isolation
  - Partition system components into trust domains
  - Combine with access controls at other layers (defense in depth)
    - Bridges are often easy to construct

- More generally, barriers can be a useful tool when systems are divided into subsystems

Communication security

- Integrity – do received bits match transmitted bits?
  - Checksums and hashes protect against accidental corruption
  - Trusted hashes can detect deliberate manipulation
- Authenticity – can recipient verify that message was sent by claimed sender?
  - Message Authentication Codes (MACs) provide authenticity if parties share a secret key
- Non-repudiation – can a third party verify that message was sent by claimed sender?
  - Digital signatures allow verification against public key (but higher overhead)

Communication security (continued)

- Confidentiality – only the intended recipient can read the message
  - Provided by encryption

- Building blocks
  - Authenticated Encryption with Associated Data (AEAD) + public key crypto
  - (Historically, block ciphers + hash functions + public key crypto)
- Challenges
  - Establish shared secret (needed by MAC, encryption)
  - Establish trust in identity (assumed by key exchange, digital signatures)
    - Avoid Man-in-the-Middle attacks, spoofing
  - Prevent replay and drop attacks – timestamps, sequence numbers, nonces
    - Avoid nonce reuse

Other encryption concepts

- Forward security
  - If shared secrets are ephemeral, compromise of private keys (long-term secrets) does not compromise confidentiality of past communications
- End-to-end encryption
  - Service provider cannot decrypt communication between two users, even when it is responsible for transporting or storing their messages
  - May not be compatible with laws in some places
  - Difficult to verify for proprietary applications

Poll: PollEv.com/cs5150

A Cornell team wants to receive data from their satellite. The signal is weak, so corruption is likely. The ground station must reject spoofed messages. Radio enthusiasts among the public are encouraged to forward data to the team (and read it if they like). The bitrate is low, so messages should be as small as possible.

Which technique would be most appropriate to deploy in this situation?

HTTPS / TLS

- Certificates (X.509) establish chain of trust
  - Pairs identity with public key, vouched by trusted authority
  - Certificate authorities validate certificates
    - Typically only ensure control of domain
  - Clients can authenticate using certificates too
- Provides **authenticity** and **confidentiality**
  - But not non-repudiation
- TLS (prev. SSL) can be used with any stream connection (e.g. TCP)
  - Must initiate handshake, verify certificate

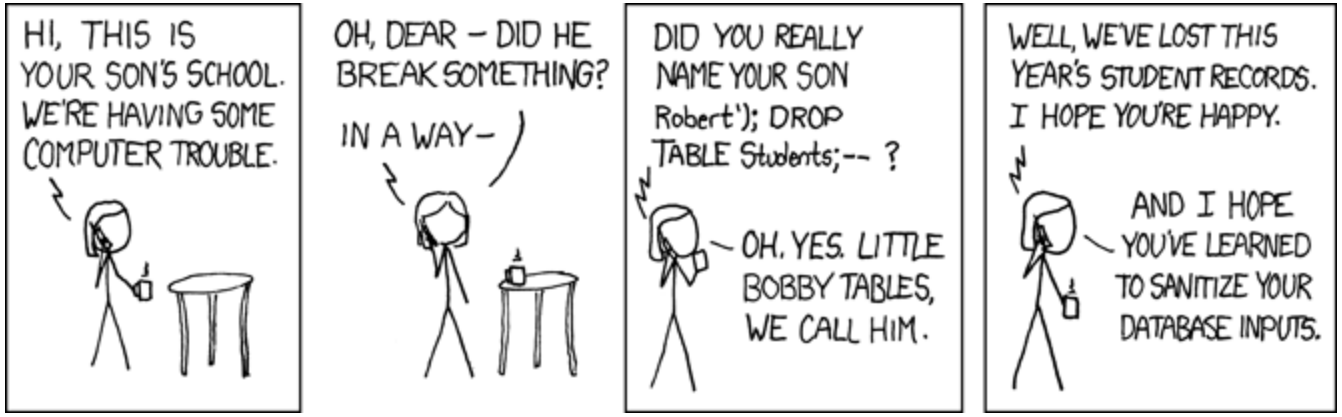
Poll: PollEv.com/cs5150

Who determines which certificates are trusted enough to show the padlock when browsing the web?

Security bugs

- Computers can execute code or commands that should not be authorized
  - Buffer overflow: user-provided bytes stored in memory are executed
  - Privilege escalation: authenticated user is able to perform operations that are only authorized for more privileged users
    - Example: system services
  - Injection: software treats user-provided data as instructions
    - Examples: printf, SQL, HTML/JS, quotes
    - Web pages are especially vulnerable (mix data and code)
      - XSS: Get a trusted page to deliver content that refers to malicious JavaScript
      - CSRF: Get authenticated web browser to submit malicious requests to a service provider

SQL injection



CC BY-NC, https://xkcd.com/327/

Command string construction

Good

- `printf("Hello, %s", name);`
-
```
var sql =
con.prepareStatement(
  "SELECT id FROM students
WHERE name = ?");
sql.setString(1, name);
sql.executeQuery();
```

Bad

- `printf("Hello, " + name);`
-
```
con.createStatement().
   executeQuery(
   "SELECT id FROM students
WHERE name = " + name);
```

Mitigating injection attacks

- Avoid splicing parameters into command strings yourself
  - Make use of placeholders and APIs accepting additional arguments
- Input validation
  - Never rely on client-side validation alone
- Taint tracking
  - Tag all values derived from user input; only allow use after passed through appropriate validation function

Side channel attacks

- Timing, power consumption, emissions can reveal secrets
- Can be mitigated with specialized techniques
  - Or occasionally by blunt ones with higher cost
  - Example: JavaScript timing precision is limited

Catching/avoiding security bugs

- Smallest mistake can compromise security
- Routinely gotten wrong by experts, large companies
- Need most advanced techniques for ensuring software quality
  - Expensive, not scalable
  - Focus community effort, leverage established components, stay up-to-date
  - Prefer simplicity (unfortunately, standards have accommodated complexity)
- Techniques:
  - Code review, expert audits
  - Testing (fuzzing)
  - Static & dynamic analysis (taint tracking, memory checking)
  - Formal verification
  - Defense in depth

Sandboxing

- Defense in depth: assume application can be compromised; how can damage be contained?
- Examples:
  - Java security manager
  - Processes vs. threads (Chrome tabs)
  - Containers and virtual machines