08 - Program Security

Acknowledgements to Dr. Ola Flygt of Växjö University, Sweden for the original slides.
Preventing security flaws

- We have seen a lot of possible security flaws. How to prevent (some of) them?
- Software engineering concentrates on developing and maintaining quality s/w
  - We’ll take a look at some techniques useful specifically for developing/maintaining secure s/w
- Three types of controls against pgm flaws:
  - Developmental controls
  - OS controls
  - Administrative controls
<table>
<thead>
<tr>
<th>Control</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developmental</td>
<td>Limit mistakes</td>
</tr>
<tr>
<td></td>
<td>Make malicious code difficult</td>
</tr>
<tr>
<td>Operating System</td>
<td>Limit access to system</td>
</tr>
<tr>
<td>Administrative</td>
<td>Limit actions of people</td>
</tr>
</tbody>
</table>
Developmental Controls: Collaborative effort

- Team of developers, each involved in ≥ 1 of stages:
  - Requirement specification
  - Design
  - Implementation
  - Testing
  - Documenting at each stage
  - Reviewing at each stage
  - Managing system development thru all stages
  - Maintaining deployed system (updates, patches, new versions, etc.)
Developmental Controls, cont.

- Fundamental principles of s/w engineering
  - Modularity
  - Encapsulation
  - Information hiding
Developmental Controls, Modularity
Developmental Controls

Encapsulation

- Minimizing information sharing with other modules
- Well documented interfaces
- “Hiding what should be hidden and showing what should be visible.”
Developmental Controls, Information hiding

- Module is a black box
- Well defined function and I/O
- Easy to know what module does but not how it does it
- Reduces complexity, interactions, covert channels, ...

=> better security
Developmental Controls, building solid software

- Peer reviews
- Hazard analysis
- Testing
- Good design
- Risk prediction & management
- Static analysis
- Configuration management
- Additional developmental controls
Developmental Controls, Peer reviews

Three types of reviews

- **Reviews**
  - Informal
  - Team of reviewers
  - Gain consensus on solutions before development

- **Walk-throughs**
  - Developer walks team through code/document
  - Discover flaws in a single design document

- **Inspection**
  - Formalized and detailed
  - Check against a prepared list of concerns
  - Statistical measures used
Developmental Controls, Hazard analysis

- Systematic exposure of hazardous system states
  - Begins Day 1
  - Continues throughout SDLC
- Tools
  - HAZOP (Hazard and operability study)
  - FMEA (Failure mode and effects analysis)
    - Bottom-up approach
  - FTA (Fault tree analysis)
    - Top-down approach
Developmental Controls, Testing

- Testing phases:
  - Module/component/unit, testing of indiv. modules
  - Integration testing of interacting (sub)system modules
  - (System) function testing checking against requirement specs
  - (System) performance testing
  - (System) acceptance testing - with customer against customer’s requirements — on seller’s or customer’s premises
  - (System) installation testing after installation on customer’s system
  - Regression testing after updates/changes to s/w
Types of testing

- **Black Box testing** - testers can’t examine code
- **White Box / Clear box testing** - testers can examine design and code, can see inside modules/system
Developmental Controls, Good design

- Good design uses:
  - Modularity / encapsulation / info hiding (as discussed earlier)
  - Fault tolerance
  - Consistent failure handling policies
  - Design rationale and history
  - Design patterns
  - Genetic Diversity
Fault-tolerant approach:
- Anticipate faults
  - (car: anticipate having a flat tire)
- Active fault detection rather than passive fault detection
  - (e.g., by use of mutual suspicion: active input data checking)
- Use redundancy
  - (car: have a spare tire)
- Isolate damage
- Minimize disruption
  - (car: replace flat tire, continue your trip)
Developmental Controls,
Good design, cont.

- Example 1: Majority voting (using h/w redundancy)
  - 3 processor running the same s/w
  - e.g., in a airplane
  - Result accepted if results of ≥ 2 processors agree

- Example 2: Recovery Block (using s/w redundancy)

<table>
<thead>
<tr>
<th>Primary Code</th>
<th>Secondary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g., Quick Sort</td>
<td>e.g., Bubble Sort</td>
</tr>
</tbody>
</table>

- Quick Sort -
  - new code (faster)
- Bubble Sort -
  - well-tested code
Developmental Controls, Good design, cont.

- Using consistent failure handling policies
- Each failure handled in one of three ways:
  - **Retrying**
    - Restore previous state, redo service using different “path”
    - e.g., use secondary code instead of primary code
  - **Correcting**
    - Restore previous state, correct some system characteristic, run service using the same code as before
  - **Reporting**
    - Restore previous state, report failure to error handler, don’t rerun service
Developmental Controls, Good design, cont.

- Using design rationale and history
  - Knowing it (incl. knowing design rationale and history for security mechanisms) helps developers modifying or maintaining system

- Using design patterns
  - Knowing it enables looking for patterns showing what works best in which situation
Developmental Controls, Good design, cont.

- Value of Good Design
  - Easy maintenance
  - Understandability
  - Reuse
  - Correctness
  - Better testing

=> translates into (saving) BIG bucks!
Developmental Controls, Risk prediction & management

- Predict and manage risks involved in system development and deployment
  - Make plans to handle unwelcome events should they occur
- Risk prediction/mgmt are esp. important for security
  - Because unwelcome and rare events can have security consequences
  - *Helps to select proper security controls* (e.g., proportional to risk)
Developmental Controls, Static analysis

- Before system is up and running, examine its design and code to locate security flaws
  - More than peer review
- Examines control flow structure, data flow structure (trail of data), data structures
- Automated tools available
  - (lint for C, FxCop for .NET programs, Perl::Critic, CheckStyle for Java)
  - Rough Auditing Tool for Security (RATS)
Developmental Controls, Configuration management

- Process of controlling system modifications during development and maintenance
  - Offers security benefits by scrutinizing new/changed code
  - Use of baseline to compare
Developmental Controls, Additional developmental controls

- Learning from mistakes
  - Avoiding such mistakes in the future enhances security
- Proofs of program correctness
  - Formal methods to verify program correctness
  - Problems with practical use of program correctness proofs
    - Esp. for large programs/systems
  - Most successful for specific types of apps
    - e.g. for communication protocols & security policies

- Even with all these developmental controls - still no security guarantees!
Operating System Controls for Security

- Developmental controls not always used

OR:

- Even if used, not foolproof
  => Need other, complementary controls, incl. OS controls

- Such OS controls can protect against some program flaws
Operating System Controls for Security,

**Trusted software**

- Trusted code establishes foundation upon which un-trusted code runs
  - Trusted code establishes security baseline for the whole system

- **OS can be trusted s/w**
Key characteristics determining if OS code is trusted

- **Functional correctness**
  - OS code consistent with specs

- **Enforcement of integrity**
  - OS keeps integrity of its data and other resources even if presented with flawed or unauthorized commands

- **Limited privileges**
  - OS minimizes access to secure data/resources
  - Trusted pgms must have “need to access” and proper access rights to use resources protected by OS
  - Untrusted pgms can’t access resources protected by OS

- **Appropriate confidence level**
  - OS code examined and rated at appropriate trust level
Operating System Controls for Security, increasing security

- Similar criteria used to establish if s/w other than OS can be trusted

- Ways of increasing security if untrusted pgms present:
  - Mutual suspicion
  - Confinement
  - Access log
Operating System Controls for Security, Mutual suspicion between programs

- Distrust other pgms - treat them as if they were incorrect or malicious
- Pgm protects its interface data
  - With data checks, etc.
Operating System Controls for Security, Confinement

- OS can confine access to resources by suspected program
  - Strict compartmentalization
    - Program can affect data and other programs only within its compartment
  - Can limit spread of malicious software
Audit log / access log

- Records who/when/how (e.g., for how long) accessed/used which objects
  - Events logged: logins/logouts, file accesses, pgm executions, device uses, failures, repeated unsuccessful commands (e.g., many repeated failed login attempts can indicate an attack)

- Audit frequently for unusual events, suspicious patterns

- Forensic measure not protective measure
  - Forensics - investigation to find who broke law, policies, or rules
Administrative Controls for Security

- They *prohibit or demand certain human behavior* via policies, procedures, etc.

- They include:
  - Standards of program development
  - Security audits
  - Separation of duties
Administrative Controls for Security,
Standards and guidelines for program development

- Capture experience and wisdom from previous projects
- Create standards and guidelines for every stage of the SDLC
Administrative Controls for Security, Separation of duties

- Break sensitive tasks into \( \geq 2 \) pieces to be performed by different people (learned from banks)

- Example 1: modularity
  - Different developers for cooperating modules

- Example 2: independent testers
  - Rather than developer testing her own code
Conclusions
(for Controls for Security)

- Developmental / OS / administrative controls help produce/maintain higher-quality (also more secure) s/w
- Art and science - no “silver bullet” solutions

“A good developer who truly understands security will incorporate security into all phases of development.”
## Summary

<table>
<thead>
<tr>
<th>Control</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developmental</td>
<td>Limit mistakes</td>
</tr>
<tr>
<td></td>
<td>Make malicious code difficult</td>
</tr>
<tr>
<td>Operating System</td>
<td>Limit access to system</td>
</tr>
<tr>
<td>Administrative</td>
<td>Limit actions of people</td>
</tr>
</tbody>
</table>