Trusting Software

The Impossible Dream Made Possible

Tim Kremann

Bottom Line

- Significant improvement is possible & affordable TODAY!

Defining Trust

- Trust is defined as the confidence one has in the software based on the available assurances that it will behave reliably and correctly while maintaining the integrity and security of itself and the system.

SW Life Cycle Examples

Software Development Phase	Examples of Benign and Intentional Errors	Examples Mitigation Strategies
Analysis & Design	Incomplete Requirements // Subtly Alter Specifications	Enforce Process
Code	Implementation Errors // Add malicious code to complex logic areas	Software Engineering Record Metrics Require Requirements to Code Traceability
Test	Fail to Run Tests on critical Functionality // Alter tests to ignore offending code	Specification Based Training Automated Testing Independent Testing
Deployment	Incorrect installation // Replace software en route	Acceptance Testing Smart Deployment Tools Tiger Teaming
Operation	Bypassing security to get work done // Hack code by using vulnerabilities	Containment and Detection Tight Configuration Control

Prongs of Trust

- High Confidence Design & Implementation
- Containment
- Detection

High Confidence D&I

- Smarter, Easier Design Tools
- Informed Testing
- Integrated Design, Implementation & Test Environment
- Disciplined Design Process
- Maintaining Trust Evidence

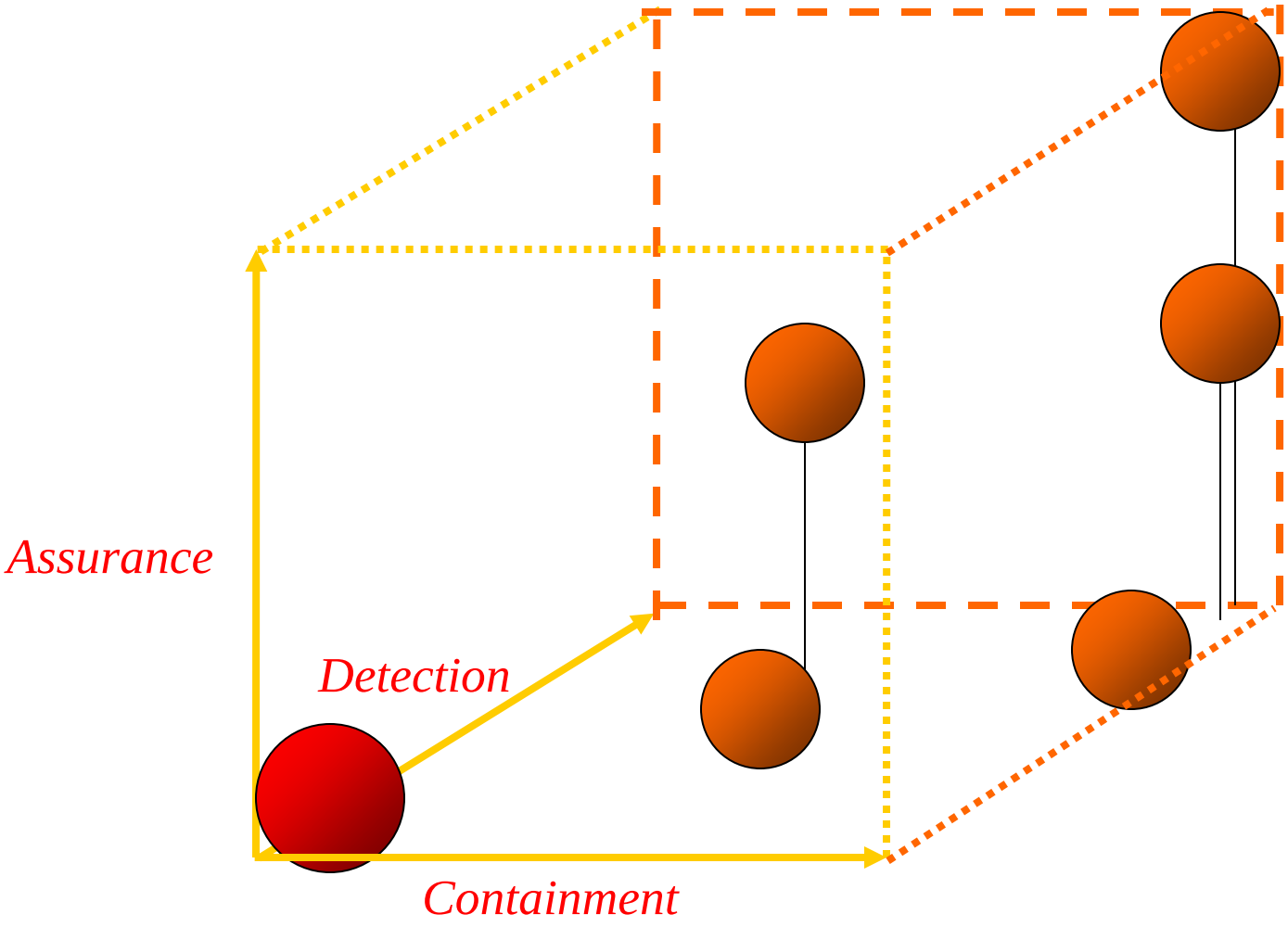
Containment

- Barriers
- Decision Rules
- Checking for “Contraband”

Detection

- Code Analysis
- Code Behavior

Trust Design Space Cube



Overall Design & Application

- Available & Emerging Techniques Imperfect
- Strength Comes from Proper Combination
- Trust Engineering Must Become Part of System Security Engineering

Recommendations

- Aim Higher
- Insist on Design & Implementation Process Improvements
- Establish Goals & Minimum Standards
- Accept Program Risk for Increased Trust