THE UNIVERSITY OF BRITISH COLUMBIA

Towards Agile Security Assurance

Konstantin Beznosov

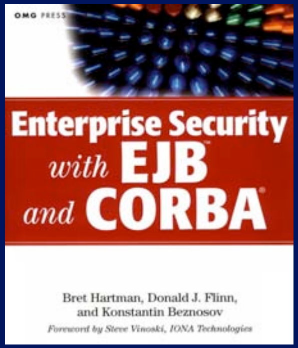
Laboratory for Education and Research in Secure Systems Engineering (LERSSE)

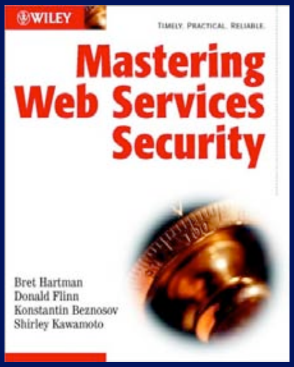
Electrical and Computer Engineering

University of British Columbia

Who's Konstantin Beznosov

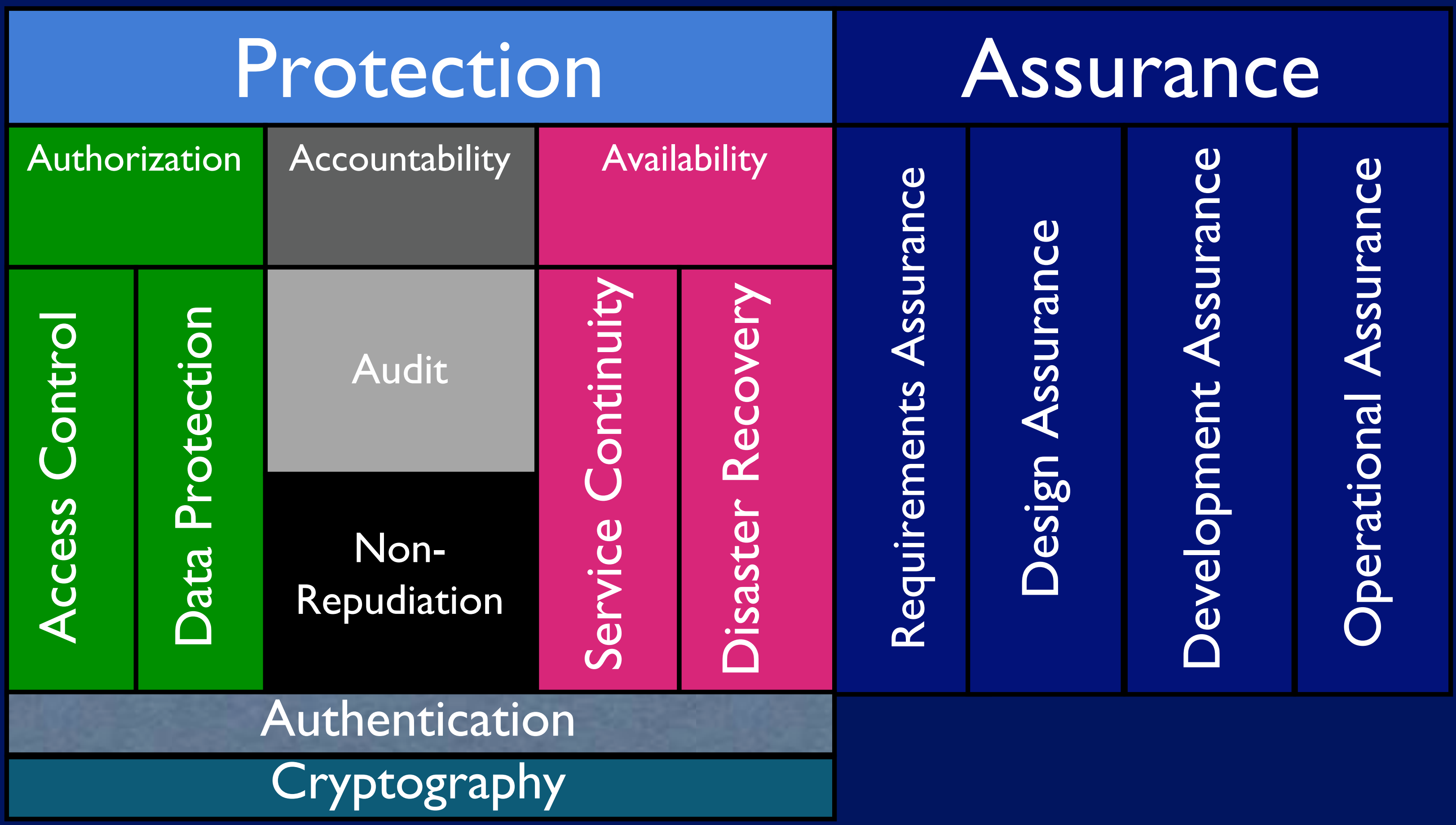
- Education
  - M.S. (1997) & Ph.D. (2000) in CS, Florida International University
  - B.S. in Physics (1993), Novosibirsk State University
- Experience
  - Assistant Prof., Electr. and Comp. Egn., UBC (2003-present)
  - Directs Laboratory for Education and Research in Secure Systems Engineering (LERSSE)
  - US industry (1997-2003): end-user, consulting, and software vendor organizations
- Contributed to
  - OMG
    - CORBA Security revisions
    - Resource Access Decision
    - Security Domain Membership Management
  - OASIS
    - eXtensible Access Control Markup Language (XACML) v1.0





Outline

- What is security and why is it hard?
- What is software security and why is it hard?
- Problem
- Contributions
- Conventional assurance & agile methods
- Solution
- Summary


Protection
Assurance
Authorization
Accountability
Availability
Access Control
Data Protection
Audit
Non-Repudiation
Service Continuity
Disaster Recovery
Requirements Assurance
Design Assurance
Development Assurance
Operational Assurance
Authentication
Cryptography

What is Security and Why is it Hard?

What is Security?

- security -- "safety, or freedom from worry"
- How can it be achieved?
  - Make computers too heavy to steal
  - Buy insurance
  - Create redundancy (disaster recovery services)

Goals of Security

- Deterrence
  - Deter attacks
- Prevention
  - Prevent attackers from violating security policy
- Detection
  - Detect attackers’ violation of security policy
- Recovery
  - Stop attack, assess and repair damage
  - Continue to function correctly even if attack succeeds
- Investigation
  - Find out how the attack was executed: forensics
  - Decide what to change in the future to minimize the risk

Solovki Monastery, White Sea, Russia

Conventional, fortress-based, security

Goal:

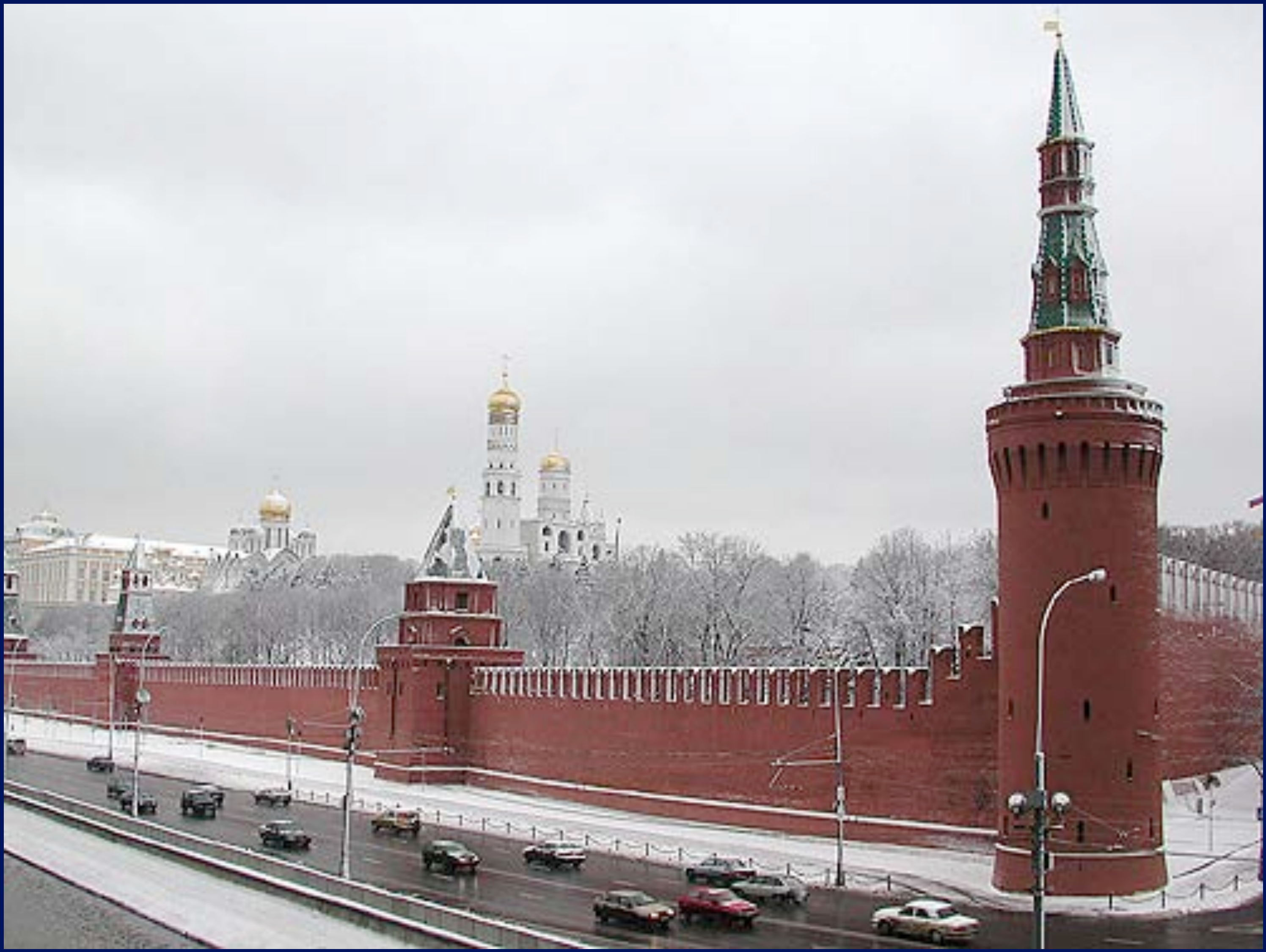
Prevent people from violating system's security policy

Means:

Fortification

- provides safety
- involves layering
- expensive
- requires maintenance
- eventually compromised

Limitations of Fortresses



Where the Fortress Analogy Breaks

Fortress

- Against external attackers
- Protects only insiders
- Defenses cannot change

Computer security

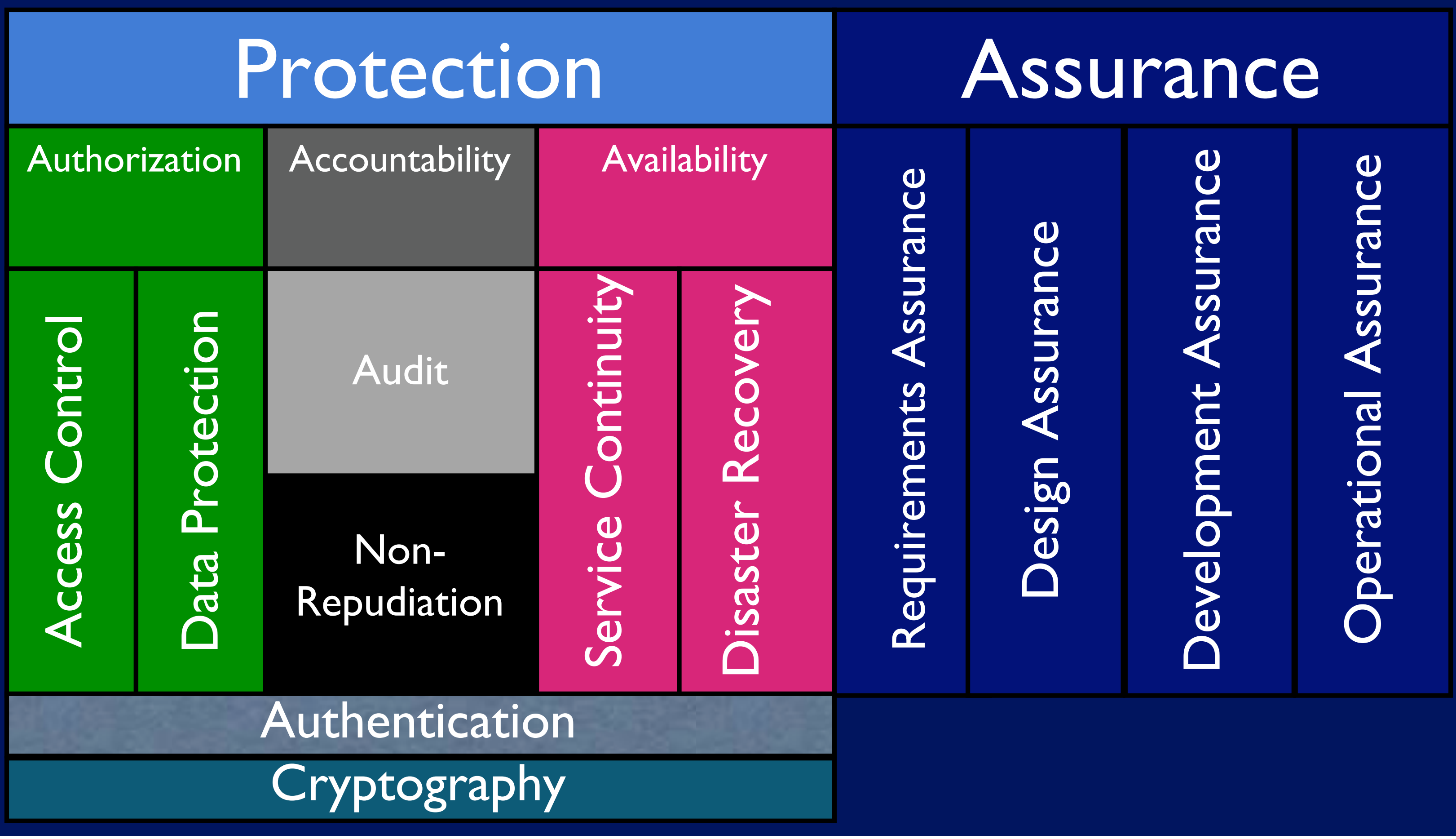
- Control of insiders
- Has to keep system usable
- Has to protect from new types of attacks

What Computer Security Policies are Concerned with?

- Confidentiality
  - Keeping data and resources hidden
- Integrity
  - Data integrity (integrity)
  - Origin integrity (authentication)
- Availability
  - Enabling access to data and resources

CIA

Conventional Approach to Security



Protection

- provided by a set of mechanisms (countermeasures) to prevent bad things (threats) from happening

Authorization

protection against breaking rules

Rule examples:

- Only registered students should be able to take exam or fill out surveys
- Only the bank account owner can debit an account
- Only hospital's medical personnel should have access to the patient's medical records

Authorization Mechanisms: Data Protection

- No way to check the rules
  - e.g., telephone wire or wireless networks
- No trust to enforce the rules
  - e.g., MS-DOS

Accountability

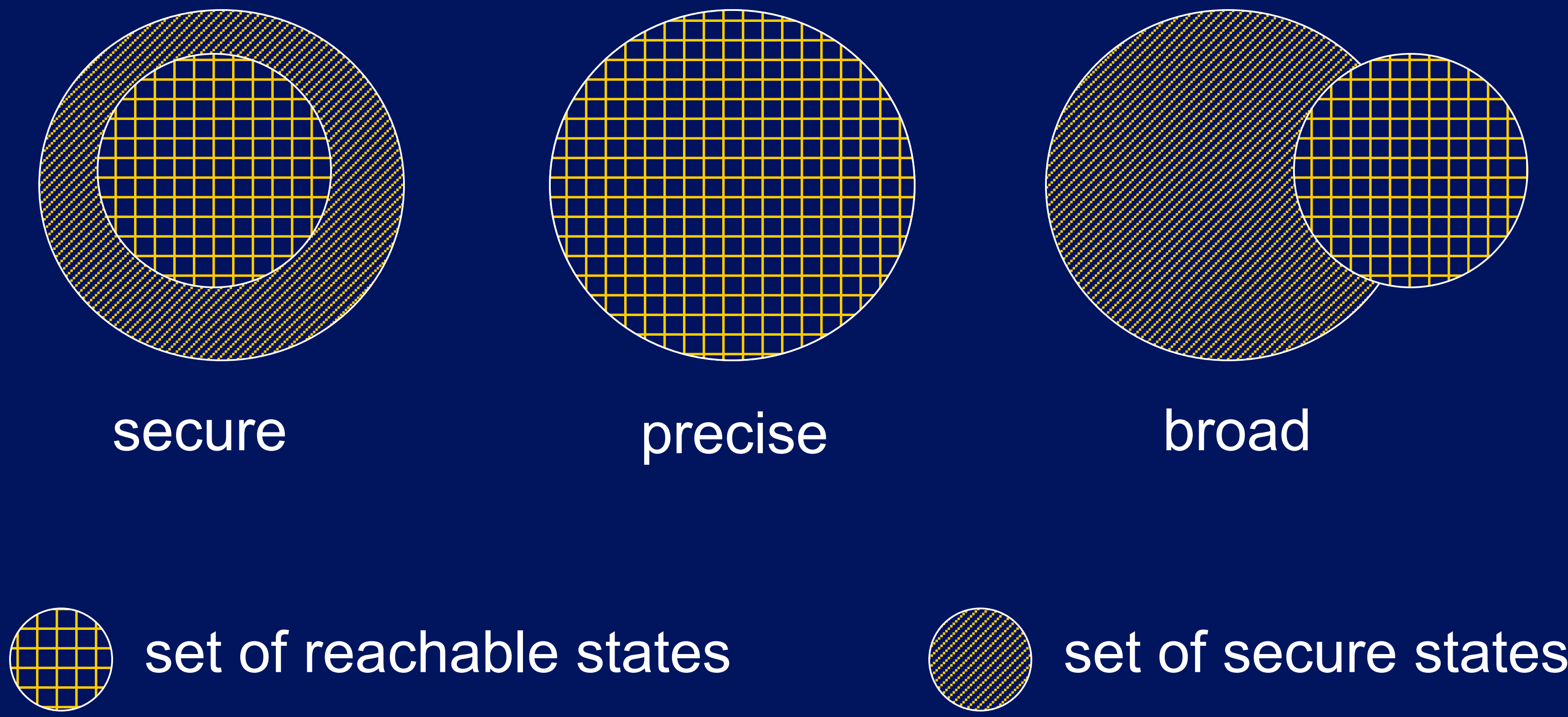
You can tell who did what when

- (security) audit -- actions are recorded in audit log
- Non-Repudiation -- evidence of actions is generated and stored

Availability

- Service continuity -- you can always get to your resources
- Disaster recovery -- you can always get back to your work after the interruption

Types of Mechanisms

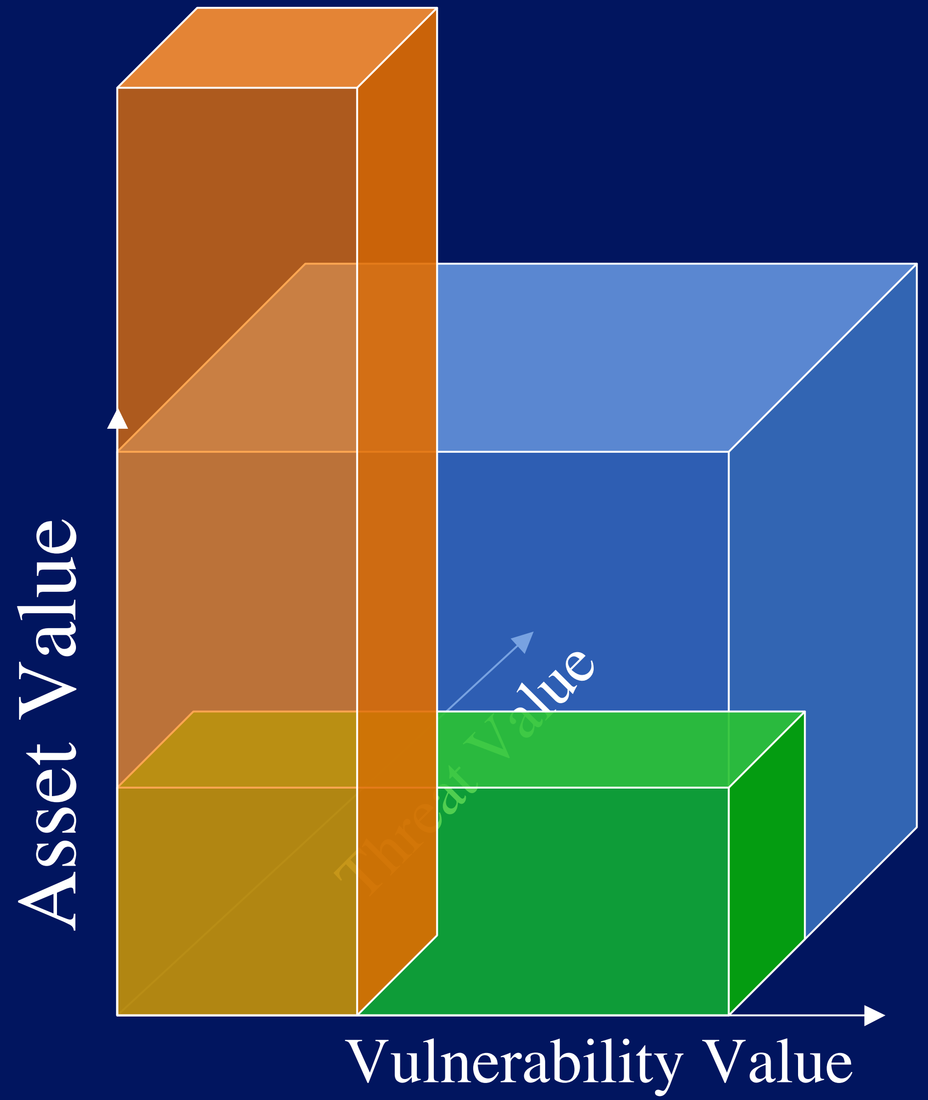


Assurance

Set of things the system builder and the operator of the system do to convince you that it is really safe to use.

- the system can enforce the policy you are interested in, and
- the system works as intended

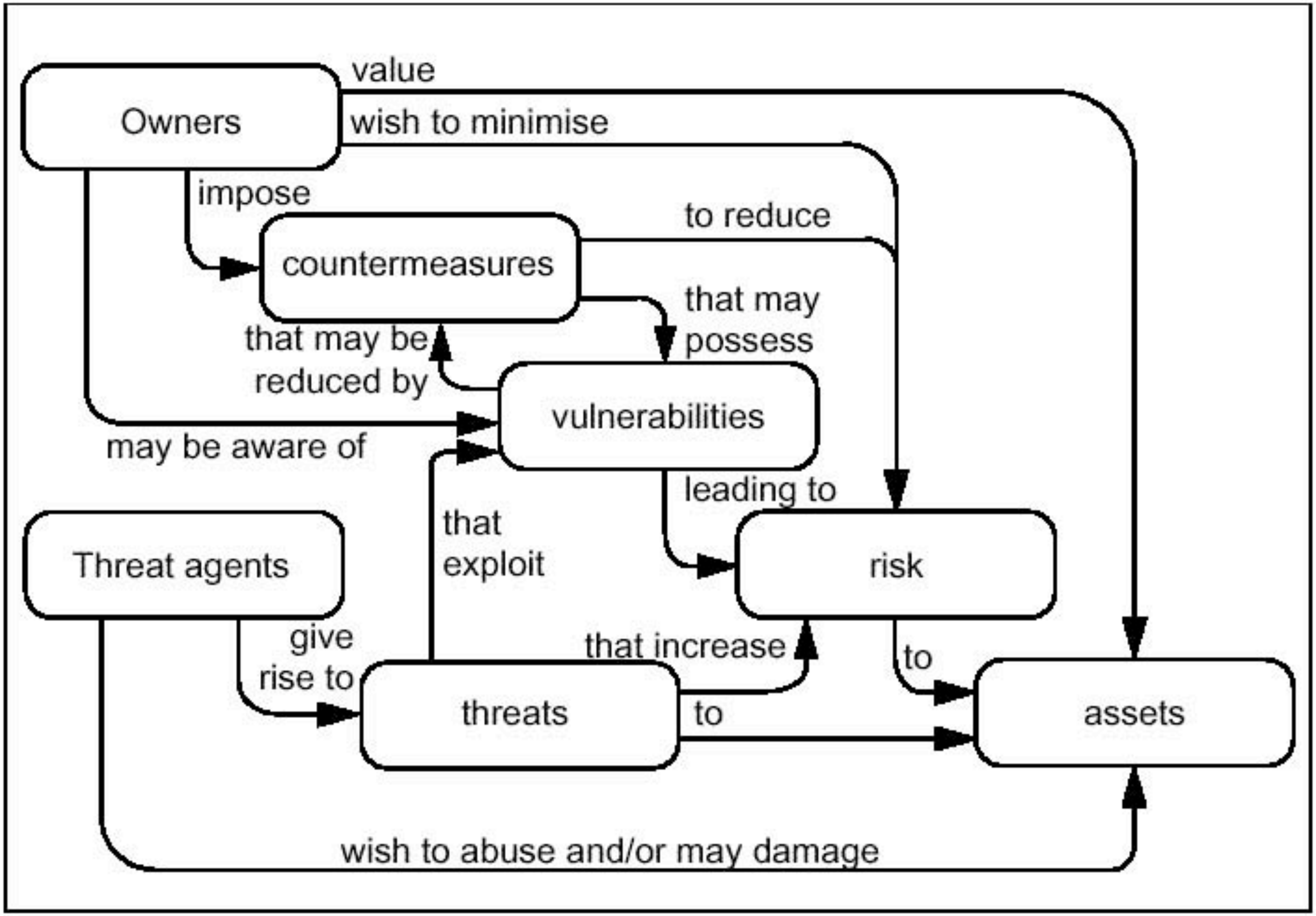
It's all about risk



Risk = Asset * Vulnerability * Threat

Classes of Threats and Means

- Disclosure
  - Snooping
- Deception
  - *Modification*
  - *Spoofing*
  - repudiation of origin
  - denial of receipt
- Disruption
  - *Modification*
  - denial of service
- Usurpation
  - Modification
  - *Spoofing*
  - Delay
  - denial of service



Source: Common Criteria for Information Technology Security Evaluation. 1999

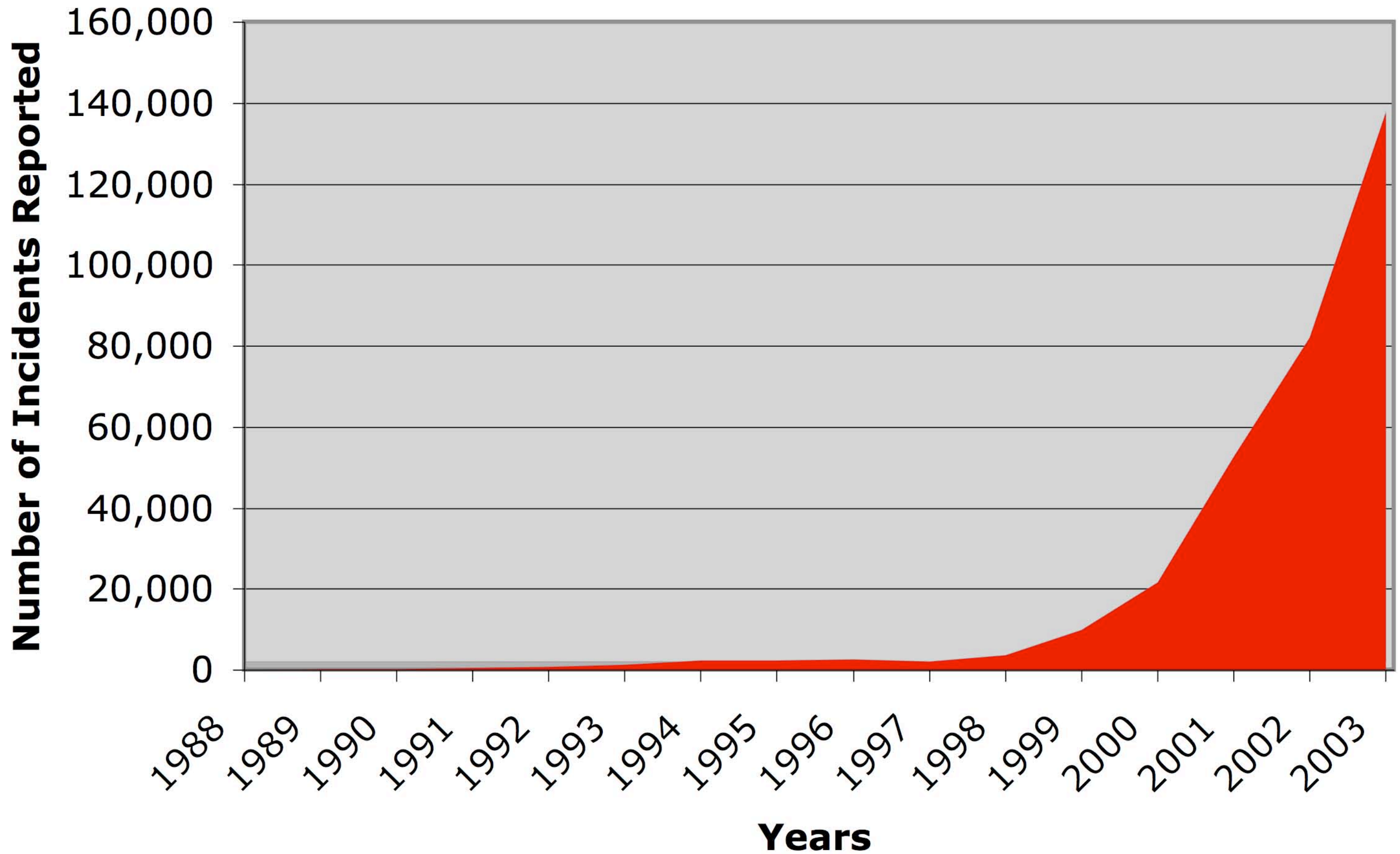
Steps of Improving Security

1. analyze risks
   - asset values
   - threat degrees
   - vulnerabilities
2. develop/change policies
3. choose & develop countermeasures
4. assure
5. go back to the beginning

THE UNIVERSITY OF BRITISH COLUMBIA

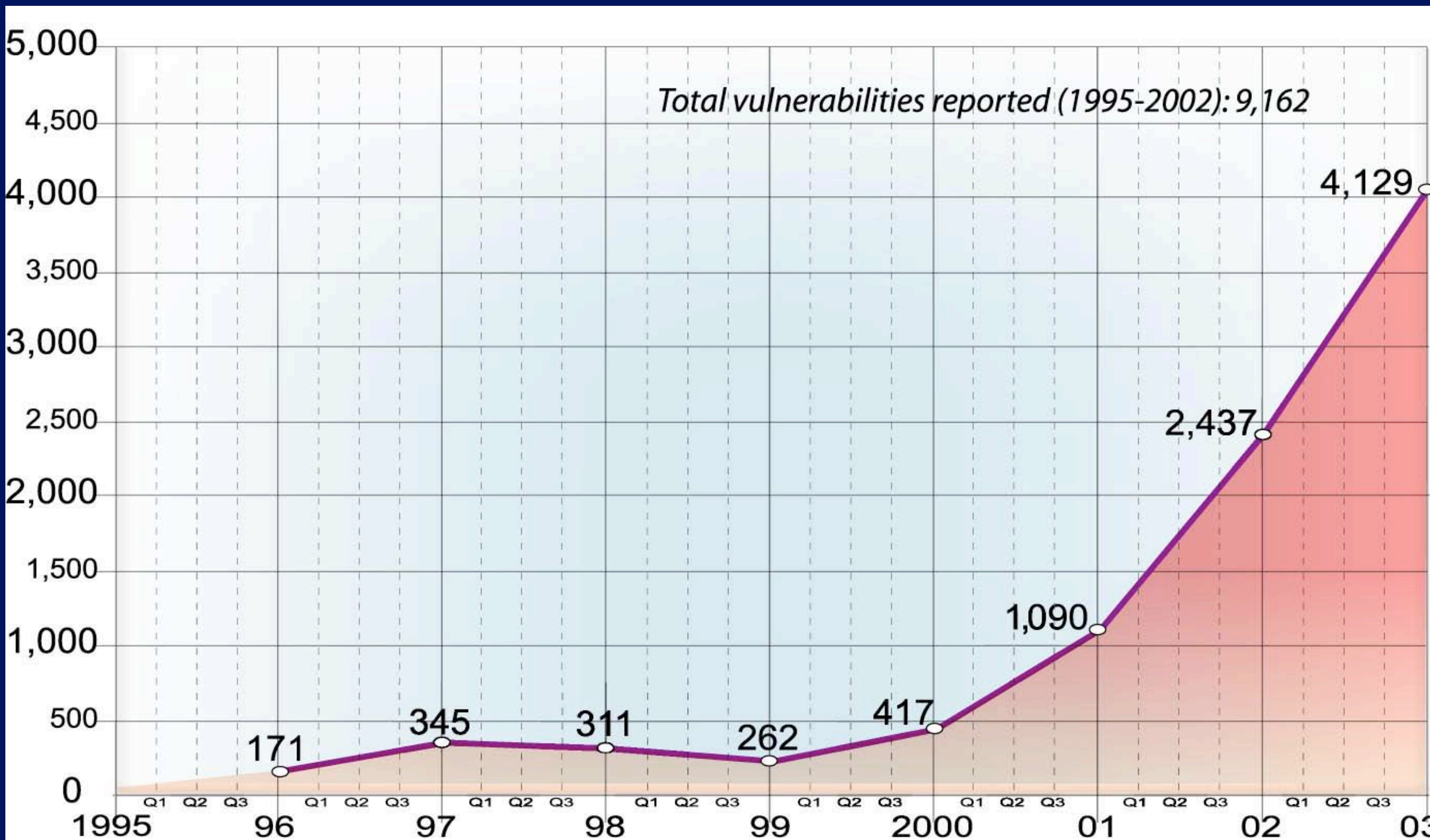
What is Software Security and Why is it Hard?

Internet security incidents reported to CERT



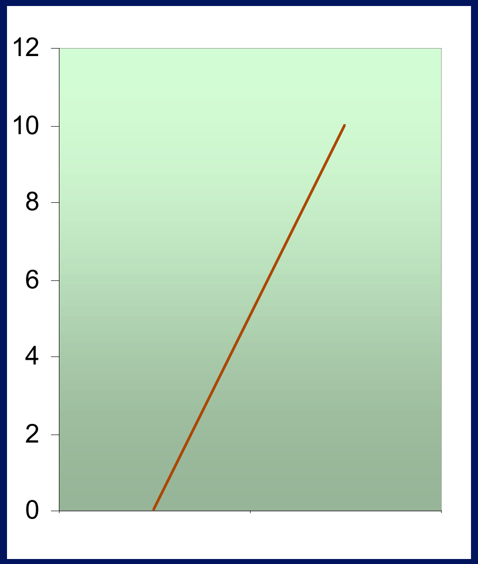
Security break-ins are all too prevalent

Vulnerability Report Statistics



Why are there so many vulnerabilities in software?

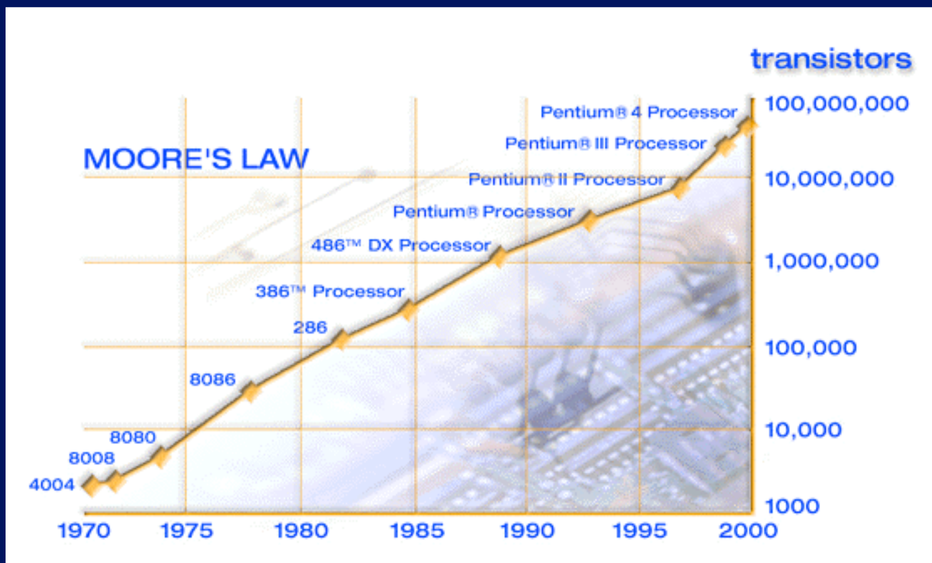
What will happen in a moment?

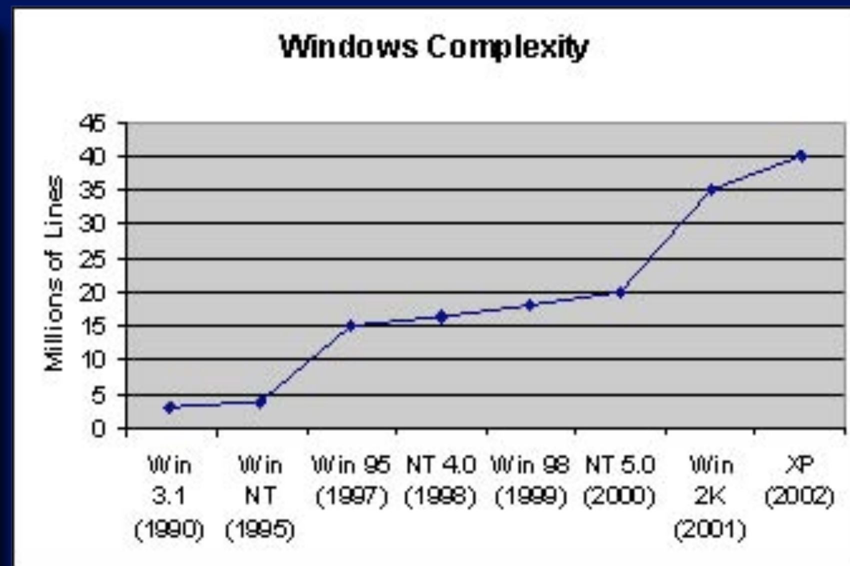


What makes simple mechanical systems predictable?

- Linearity (or, piecewise linearity)
- Continuity (or, piecewise continuity)
- Small, low-dimensional statespaces

Systems with these properties are
(1) easier to analyze, and (2) easier to test.



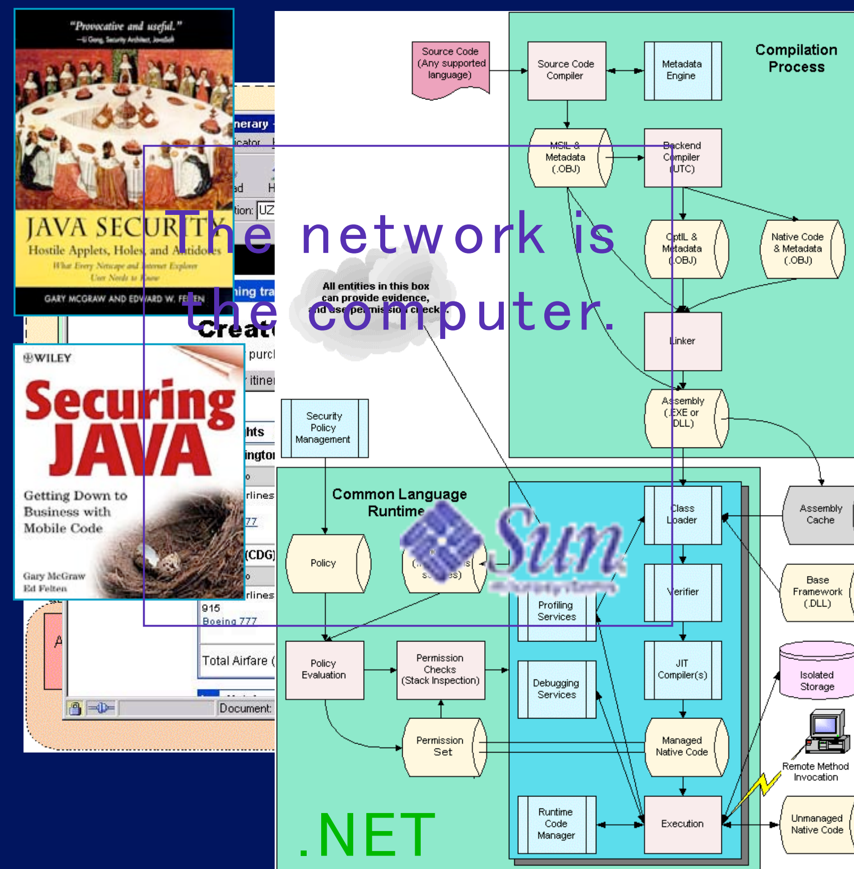


- Computers enable highly complex systems
- Software is taking advantage of this
  - Highly non-linear behavior; large, high-dim. state spaces

Other software properties make security difficult

The Trinity of Trouble

- **Connectivity**
  - The Internet is everywhere and most software is on it
- **Complexity**
  - Networked, distributed, mobile, feature-full
- **Extensibility**
  - Systems evolve in unexpected ways and are changed on the fly

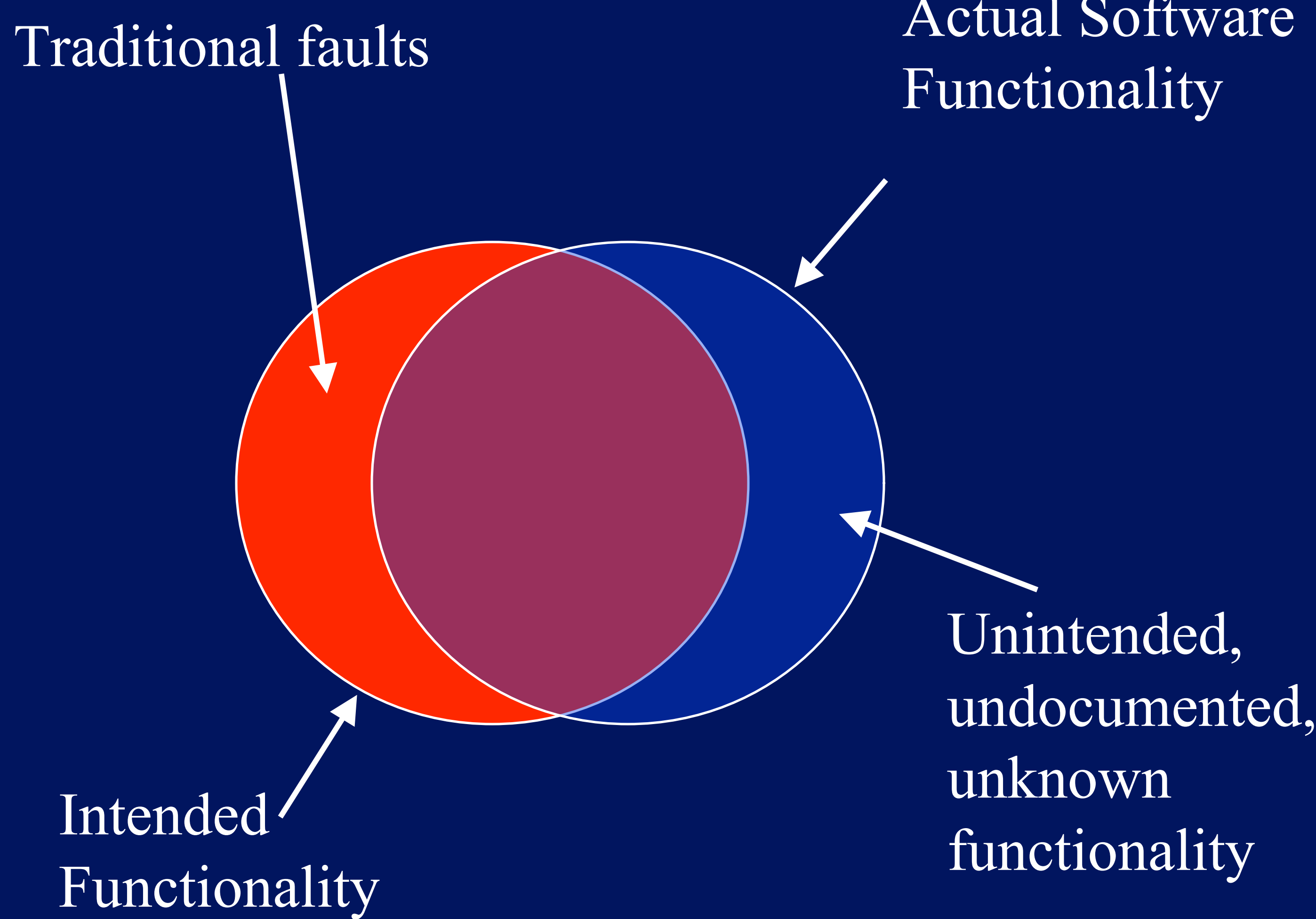


How Are Security Bugs Different?

When is a security bug not like a bug?

- Traditional non-security bugs -- often defined as a violation of a specification.
- Security bugs -- additional behavior, not originally intended
  - Meanwhile, it is doing what it is supposed to do
  - Traditional techniques not good at finding
  - Even in inspections, tend to look for
    - missing behavior
    - incorrect behavior
  - Neglect to look for ... undesirable side-effects

Intended vs. Implemented Behavior



Traditional faults

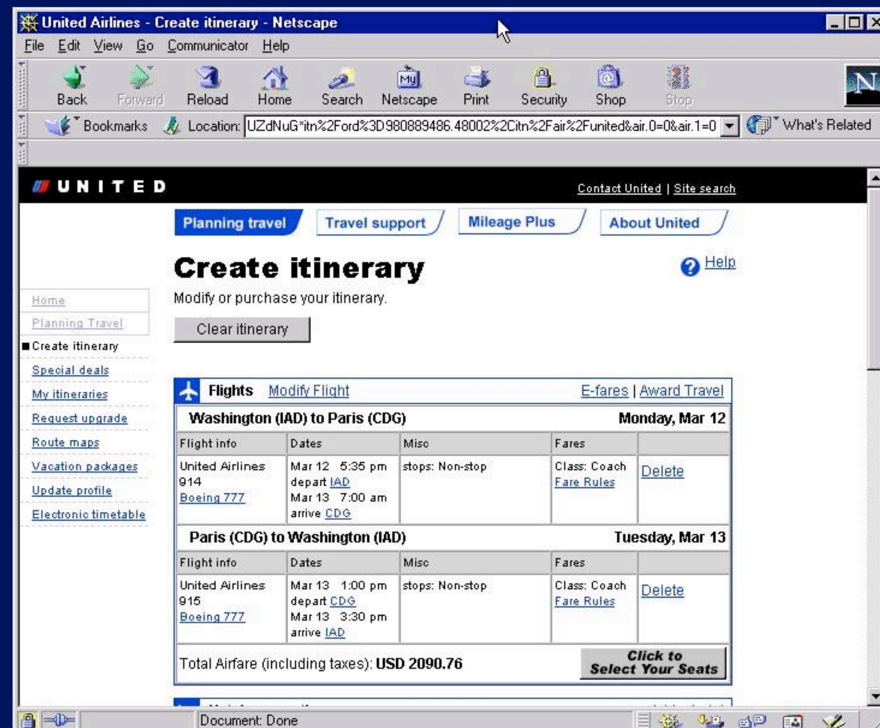
- Incorrect
  - Supposed to do A but did B instead
- Missing
  - Supposed to do A *and* B but did only A.

Security Bugs

- Side effects
  - Supposed to do A, and it did.
  - In the course of doing A, it *also* did B
- Monitoring for side effects and their impact on security can be challenging
  - Side effects can be subtle and hidden
  - Examples: file writes, registry entries, extra network packets with unencrypted data

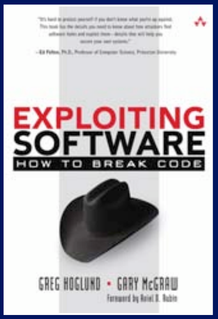
Attack pattern examples

- Exploit race condition
- Provide unexpected input
- Bypass input validation



49 Types of Software Attacks

1. Make the Client Invisible
2. Target Programs That Write to Privileged OS Resources
3. Use a User-Supplied Configuration File to Run Commands That Elevate Privilege
4. Make Use of Configuration File Search Paths
5. Direct Access to Executable Files
6. Embedding Scripts within Scripts
7. Leverage Executable Code in Nonexecutable Files
8. Argument Injection
9. Command Delimiters
10. Multiple Parsers and Double Escapes
11. User-Supplied Variable Passed to File System Calls
12. Postfix NULL Terminator
13. Postfix, Null Terminate, and Backslash
14. Relative Path Traversal
15. Client-Controlled Environment Variables
16. User-Supplied Global Variables (DEBUG=1, PHP Globals, and So Forth)
17. Session ID, Resource ID, and Blind Trust
18. Analog In-Band Switching Signals (aka “Blue Boxing”)
19. Attack Pattern Fragment: Manipulating Terminal Devices
20. Simple Script Injection
21. Embedding Script in Nonscript Elements
22. XSS in HTTP Headers
23. HTTP Query Strings
24. User-Controlled Filename
25. Passing Local Filenames to Functions That Expect a URL
26. Meta-characters in E-mail Header
27. File System Function Injection, Content Based
28. Client-side Injection, Buffer Overflow
29. Cause Web Server Misclassification
30. Alternate Encoding the Leading Ghost Characters
31. Using Slashes in Alternate Encoding
32. Using Escaped Slashes in Alternate Encoding
33. Unicode Encoding
34. UTF-8 Encoding
35. URL Encoding
36. Alternative IP Addresses
37. Slashes and URL Encoding Combined
38. Web Logs
39. Overflow Binary Resource File
40. Overflow Variables and Tags
41. Overflow Symbolic Links
42. MIME Conversion
43. HTTP Cookies
44. Filter Failure through Buffer Overflow
45. Buffer Overflow with Environment Variables
46. Buffer Overflow in an API Call
47. Buffer Overflow in Local Command-Line Utilities
48. Parameter Expansion
49. String Format Overflow in syslog()



THE UNIVERSITY OF BRITISH COLUMBIA

clash between assurance & agility

Problem

Mismatch between

- agile methodologies for software development
- conventional methods for security assurance

Hard to assure with agile development

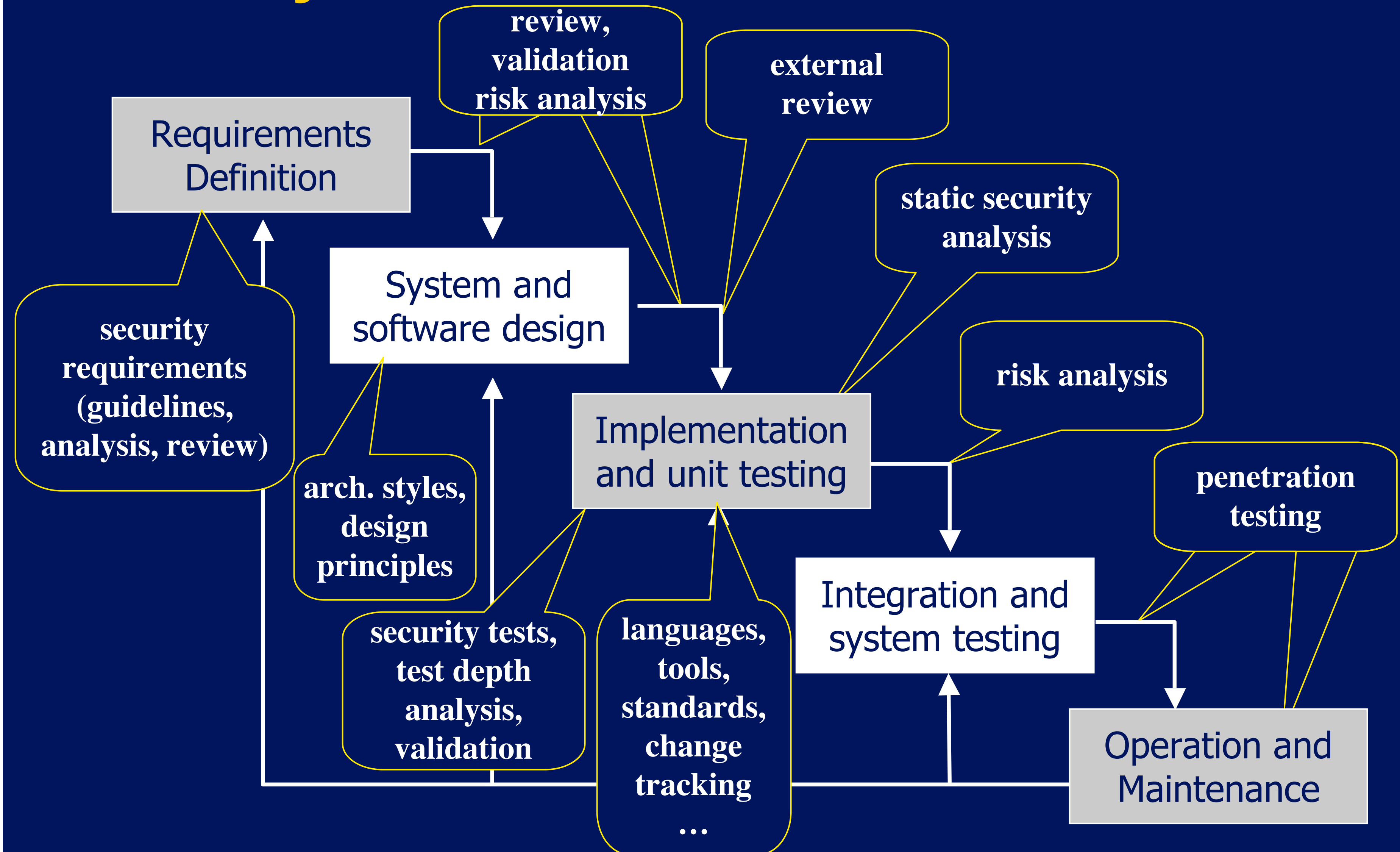
Why is addressing the mismatch **important?**

- More security-critical software
- Agile methods are there to stay

Contributions

1. examined the mismatch between security assurance and agile methods
2. classified conventional security assurance practices according to the degree of clash
3. suggested ways of alleviating the conflict

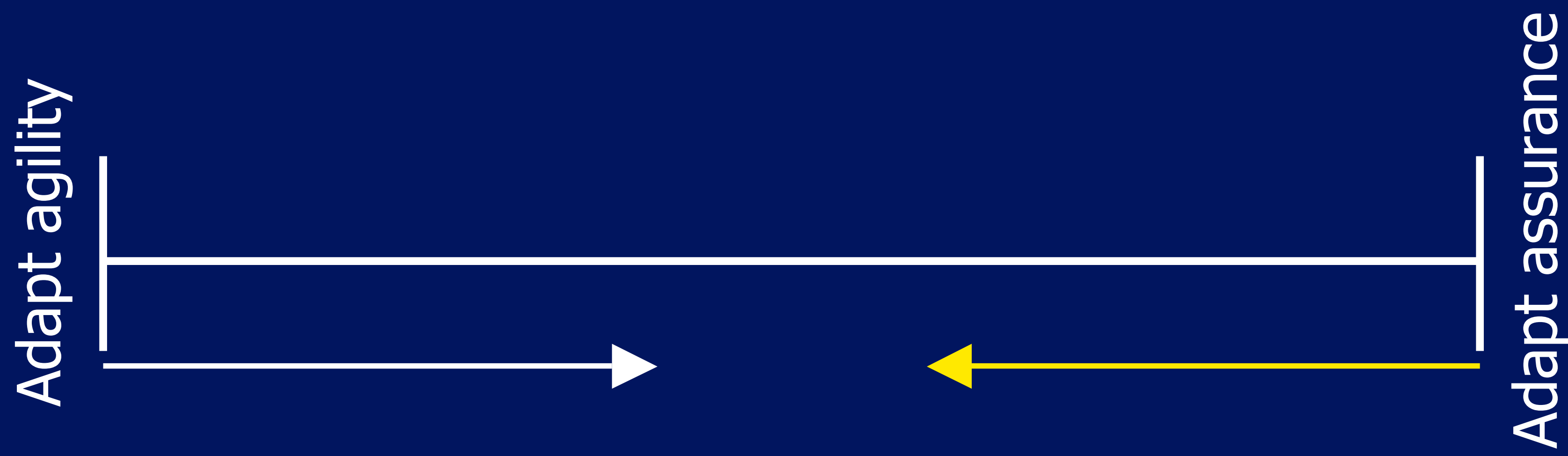
What's Conventional Security Assurance for Software is About?



Adapted from
D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security & Privacy*, vol. 2, no. 4, 2004, pp. 79-84.

Solution(s)?

If the mountain will not go to Mahomet,
let Mahomet go to the mountain. (proverb)



Examination Results

Assurance relies on third party

- reviews
- evaluation
- testing

Points of clash

1. direct communication and tacit knowledge
2. iterative lifecycle
3. design refactoring
4. testing “philosophy”

(Mis)match Classification

1. Natural Match

   e.g., XP pair programming ♥ internal review & coding standards

2. Methodology-neutral

   e.g., language (e.g., Java, C# vs. C, C++),
   version control and change tracking

3. Can be (semi-)automated

   e.g., code static analysis,
   security testing/scanning

4. Mismatch (≈ 50%)

   e.g., external review, analysis,
   testing, validation change authorization

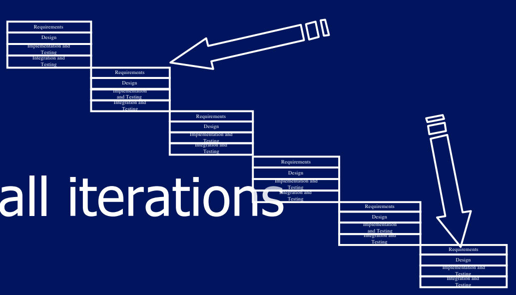
Alleviating the Mismatch

For (semi)-automatable

- Increase acceptance through **tools**
- Codify security knowledge in tools
  - automated fault injection, test generation

For mismatching

- Search for new agile-friendly assurance methods
  - direct communication and tacit knowledge
  - iterative lifecycle
  - design refactoring
  - testing “philosophy”
- Intermittent assurance
  - apply at the first and last iterations
  - use the results to “align” the development
  - Have a security engineer (role) involved in all iterations (Wäyrynen et al. 2004)



Summary

Problem

mismatch between agile development & security assurance

Contributions

1. **Examined** (pain points)
2. **Classified** assurance methods
3. **Alleviated** (tools, knowledge codification, new methods research, intermittent assurance)

THE UNIVERSITY OF BRITISH COLUMBIA

konstantin.beznosov.net

K. Beznosov and P. Kruchten, "Towards Agile Security Assurance," in *Proceedings of The New Security Paradigms Workshop*, White Point Beach Resort, Nova Scotia, 20-23 September 2004. pp. 47-54.