Software Reliability and Security

Module 9

Winter 2017

Outline

- Security Types
- Computer Attacks and Defenses
- Attack Defense - Intrusion detection systems and testing
- Security Engineering
- Software Engineering for Security

Presentation/Lecture Schedule and Report Due Dates

- Presentation 1
  - Related background paper
  - Jan 27, Feb 1, 3
- Presentation 2
  - Project proposal
  - March 1, 3, 8
- Presentation 3
  - Final project report
  - March 24, 29, 31

- Lectures
  Jan 13, 18, 20, 25, 27
  Feb 1, 3, 8, 10, 15, 17
  March 1, 3, 8, 10, 15, 17, 22, 24, 29, 31

- Project Proposal Due
  Tuesday, February 28

- Final Project Report Due
  Monday, April 10

- Final Exam
  Wednesday, April 12, 10:00am

Security Engineering

- What is Security Engineering? (Anderson 2001)
  - Builds systems to remain dependable in the face of malice, error or mischance
- Security Engineering Life Cycle (ISO/IEC 15288)
  - Concept stage
  - Development stage
  - Production stage
  - Utilization stage
  - Support stage
  - Retirement stage

– most common in other engineering disciplines

Stakeholders of Security Engineering

- Developers
- Product vendors
- Integrators
- End users or customers
- Security evaluation/certifying organizations
- System administrators
- System maintenance /monitoring service providers

Interactions with Other Disciplines

- Main challenge in security engineering - requires cross-disciplinary expertise and system engineering skills
  - Enterprise engineering
  - Systems engineering
  - Software engineering
  - Hardware engineering
  - Human factors engineering
  - Communications engineering

Security Engineering Sub-Discipline

- Operations Security
- Information Security
- Network Security
- Physical Security
- Personnel Security
- Administrative Security
- Communications Security
- Emanation Security
  - Deals with undesired signals generated by machines that can transmit information outside the security domain
- Computer Security

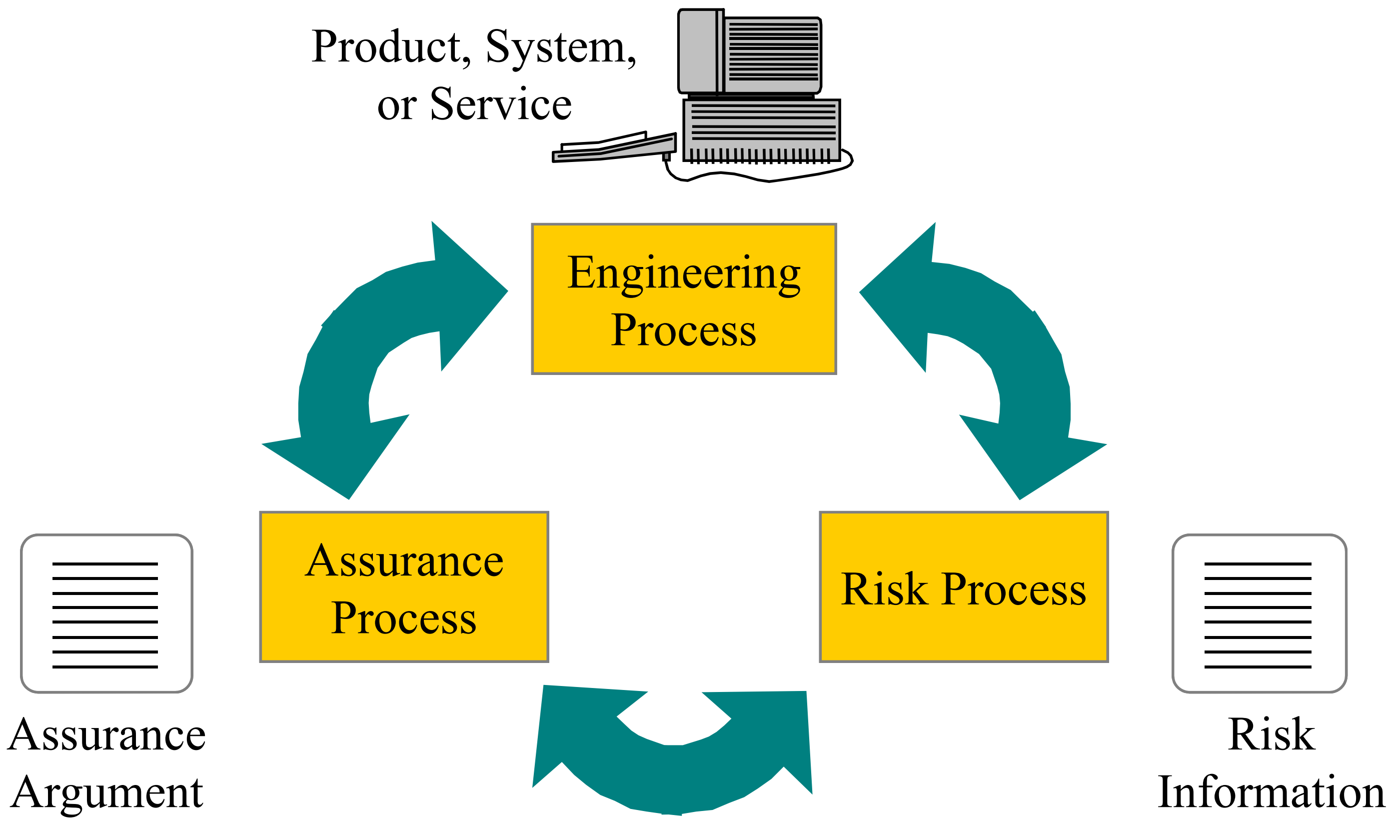
Security Engineering Goals

- Identify the security risks
- Identify security needs based on risks
- Transform security needs into security guidance
- Ensure the effectiveness of the security guidelines
- Determine if the impacts due to residual security vulnerabilities are tolerable
- Integrate other disciplines to measure the trustworthiness of the whole system

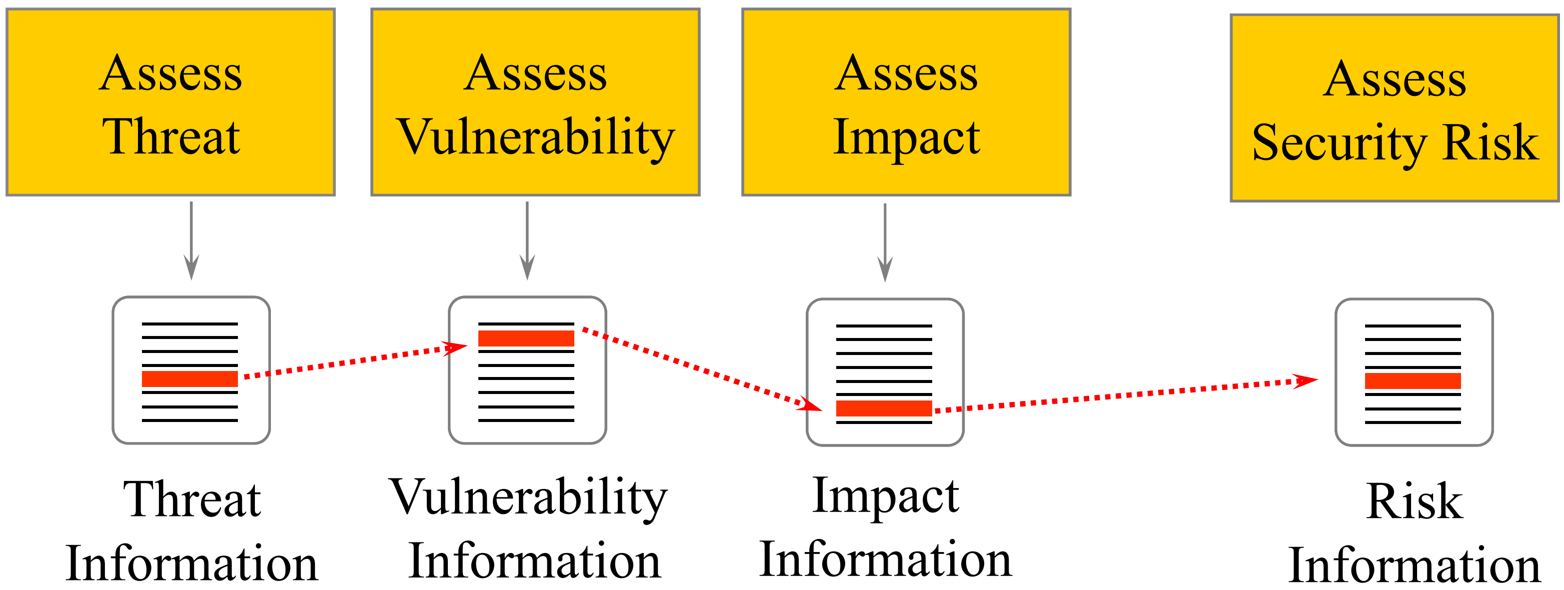
Security Engineering Process Overview

- Three Basic Areas
  - Risk Process – Identify, prioritize, and manage the unwanted incidents (threat, vulnerability, impact)
  - Engineering Process – Works with the other engineering disciplines to determine and implement solutions to mitigate the identified risks
  - Assurance Process – Establishes confidence in the security solutions
    - What is security?
    - What is confidence?
    - How can we measure?

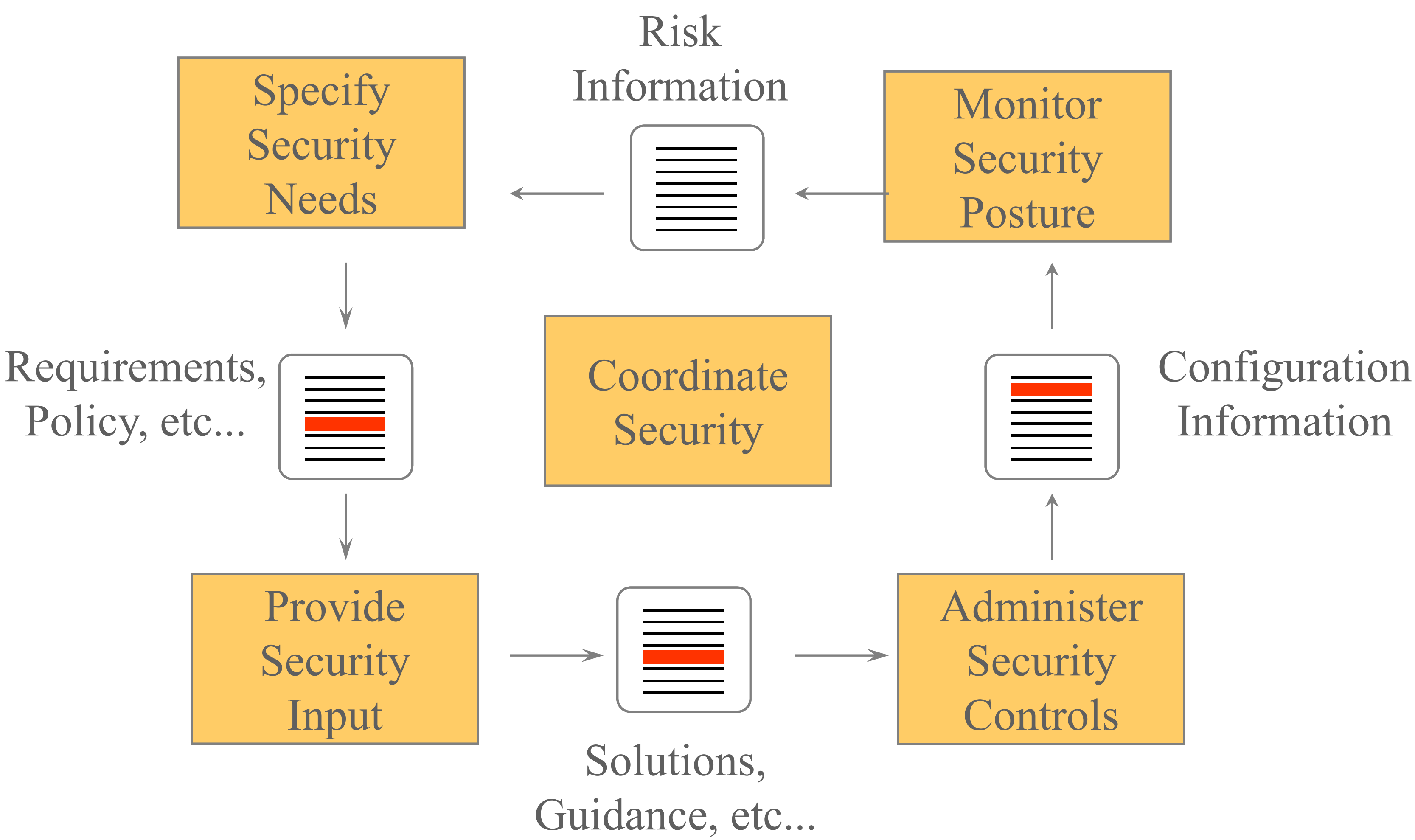
Security Engineering Process Overview



Risk Process

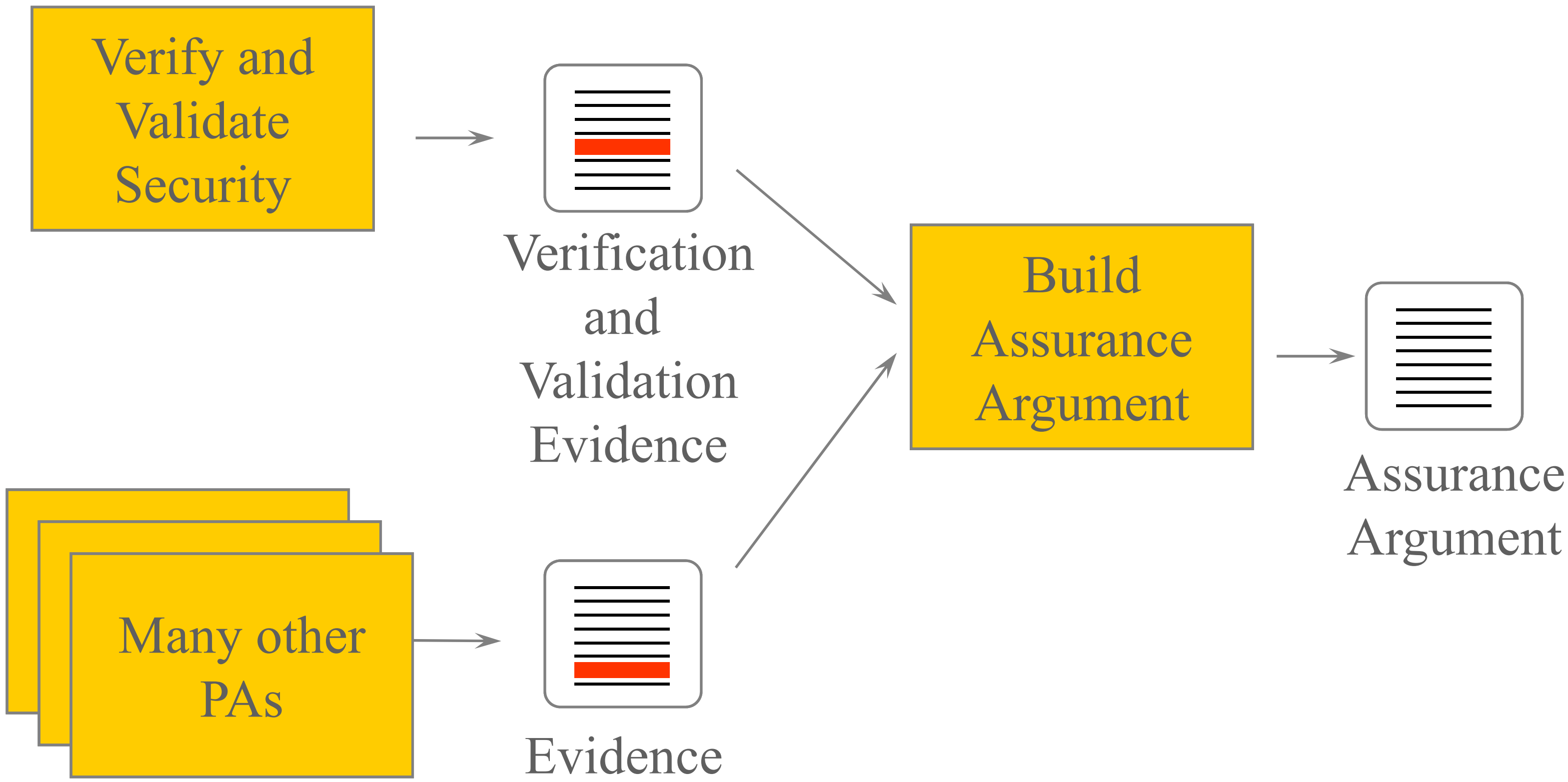


Engineering Process



Assurance Process

The degree of confidence that security needs are satisfied



Software Engineering for Security

- Software engineers should think about both users & attackers
- Software engineering objectives
  - Functionality, usability, efficiency, etc.
- Security engineering objectives
  - Privacy, confidentiality, integrity, availability, prevention, traceability , auditing, and monitoring
- Software security engineering objectives
  - Meet both security and software project goals

Is Security Engineering Really Just Good Software Engineering (Wolf 2004)?

- While software engineering is about ensuring that certain things happen, security is about ensuring that they don't (Anderson, 2001)
- A security failure results from an attack that exploits a vulnerability, where a vulnerability can be viewed as a fault
- One major difference
  - Software engineering assumes (like traditional engineering)
    - Rare events are truly rare
    - Tradeoffs (cost vs. quality vs. performance vs. …) based on this assumption
  - Security engineering is all about rare events
    - Basis for success of adversaries
- Security engineering is indeed "just" good (not traditional) software engineering

A Software Risk Management Process

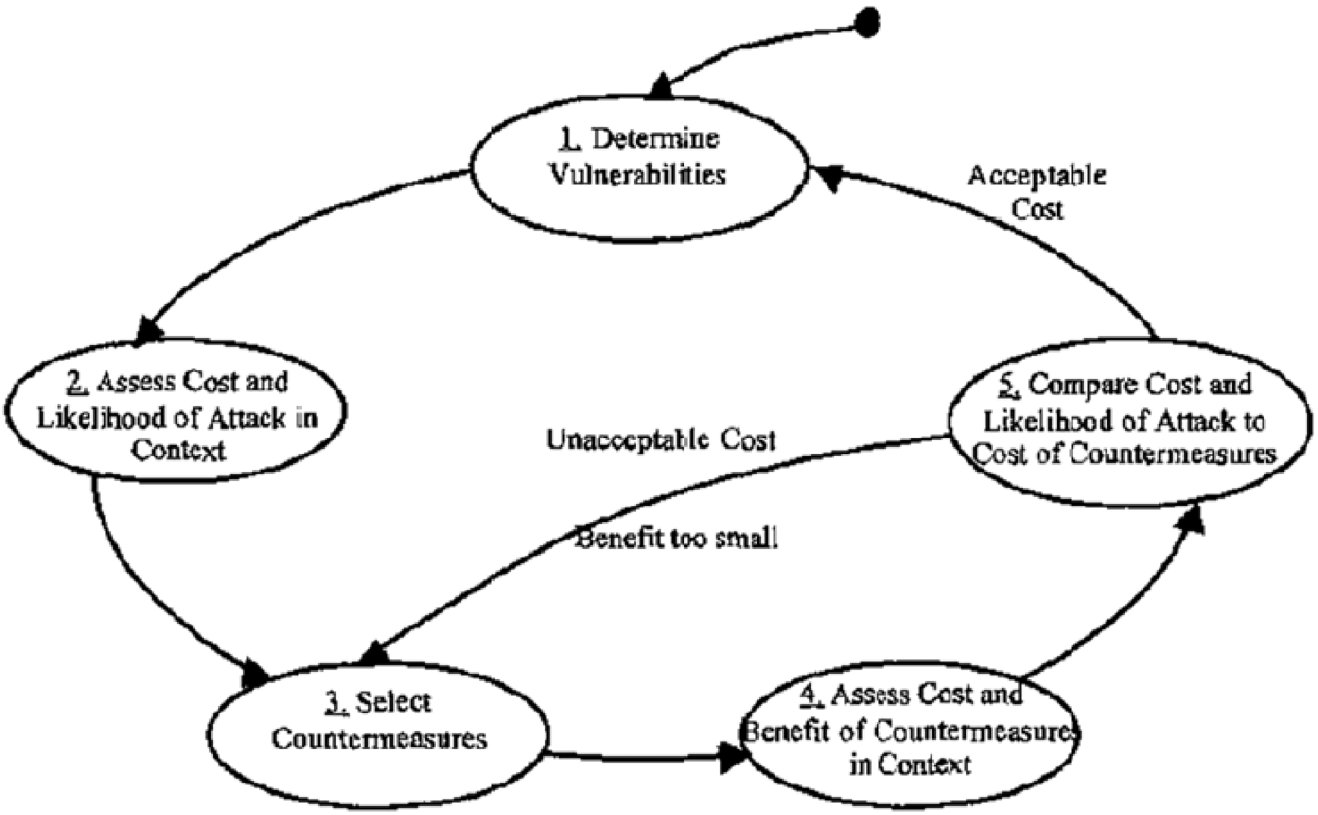
- Spiral Model – from security view point only
  - In spiral model, risk = security risk
  - Consider security when requirements are derived
  - Prioritize security risks and evaluate strategies for addressing those
  - Develop prototype and validate that the solution addresses security risks
  - Integrate the solution into the current artifacts (code, design, and requirements)
  - Plan for the next phase

Role of Software Security Personnel in Life Cycle

- Deriving Requirements
  - Define software requirements using as formal as possible
  - Identify what needs to be protected, from whom and for how long
  - Classify/prioritize software security risks
- Risk Assessment
  - Rank the potential security risks based on their severity from expert knowledge and specification
- Design for Security
  - Data flow between components, component roles, trust relationships
- Secure Implementation
  - Review code (code auditing) and document
- Security Testing
  - Based on previously ranked set of potential risks
  - Test a system in an environment similar to an actual one

Appropriate and Effective Guidance in Information Security (AEGIS)

- Based on Boehm's spiral model and Viega and McGraw's spiral model for security



Software Evaluation Checklist for Security

- How will the application interact with its environment?
- Users/administrators - external or internal or both?
- What protocols will be used (communicating components traffic directions, ports opened)?
- Encrypted or not, type of encryption mechanism and key management?
- Consistent with current network security configurations and policies (proxy servers, firewalls, etc.)?
- Security is an integral part of the product, or an after thought?
- Release a patch when a security flaw is discovered?
- Has security recommendations (provides/recommends a default architecture)?

Software Specification Languages

- Requirements Specification Language
  - Expresses what the system should do - not how it should do it (this separation is not always clear)
  - Also called functional specification language (usually used to specify expected functions)
  - A non-functional system requirement is a restriction or constraint placed on a system service (e.g., response time)
  - Example – UML, SDL, MSC, Promela, AsmL
- Objectives
  - To define data to be processed
  - To describe behavior or functions
  - To specify performance requirements
  - To apply appropriate verification/validation process

Software Specification Languages – contd.

- Types of Specification Languages
  - State-oriented
  - Object oriented
  - Process oriented
  - Executable/Non-executable
  - Formal/Informal
  - Visual/Non-Visual
  - …
- Essential Characteristics of a Specification Language
  - Complete - suitable to express all the requirements
  - Unambiguous- well-defined syntax and semantics
  - Abstract – focus on interesting aspects and allow separation of concerns

Unified Modeling Language (UML)

- Standardized by Object Management Group (OMG)
- Specify, visualize, and document models of software systems, including their structure and design
- Defines diagrams under three categories
  - Structural Diagram (static application structure)
    - Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram
  - Behavior Diagrams (aspects of dynamic behavior)
    - Use Case Diagram, Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram
  - Model Management Diagrams (Organize and manage application modules)
    - Packages, Subsystems, and Models
- More Information - http://www.uml.org/

Attack Scenario Description - Attack Languages

- Encode the manifestations of an attack in a suitable format
- Recognize an attack given a manifestation
- React to or report an attack
- Analyze the relationships among different attacks to identify coordinated attacks
- Describe attack histories/scenarios for reproducing attacks for testing

Attack Languages Objectives

- Simplicity - provide features only to represent attack scenarios
- Expressiveness – represent any attack signature that is detectable
- Rigor – implementation-independent syntax and semantics for unambiguous attack description
- Extensibility – extend the language for new domains (new event types)
- Executability / Translatability – automatically incorporate attack descriptions to an IDS
- Portability – adaptable to different environments
- Heterogeneity – describe attacks using events from multiple domains

Classification of Attack Languages

- Event Languages
  - Describe events mainly based on the specification of data format
  - E.g.,: BSM (Basic Security Module) audit record specs, TCPDump packets
- Response Languages
  - Specify the actions to be taken in response to the detection of attack
  - Usually use library functions of programming language such as C, Java
- Reporting Languages
  - Describe alerts about an attack (e.g., source, target, and type of attack, related events)
  - Examples: Common Intrusion Specification Language (CISL), Intrusion Detection Message Exchange Format (IDMEF)

Classification of Attack Languages – contd.

- Correlation Languages
  - Specify relationships among attacks to identify coordinated intrusion attempts
  - Examples: Honeywell's ARGUS, UCSBs STATL (event based)
- Exploit Languages
  - Describe the steps to be followed to perform an intrusion
  - Usually use programming languages (C, C++, Perl, …).
- Detection Languages
  - Matches patterns at run-time against observed events to detect intrusions
  - Examples: Bro, Snort

Example Attack Languages

- State Transition Analysis Technique Language (STATL)
  - Attacks are described as sequences of actions - attack scenarios
  - STATL is an attack scenario description language
  - The monitored system is represented as a state transition diagram
  - A transition takes place on some Boolean condition being true
  - The guard conditions filter intrusive activities from non-intrusive ones
- SecureUML
  - Model access control policies based on role based access control (RBAC) and integrate it into a model-driven development process

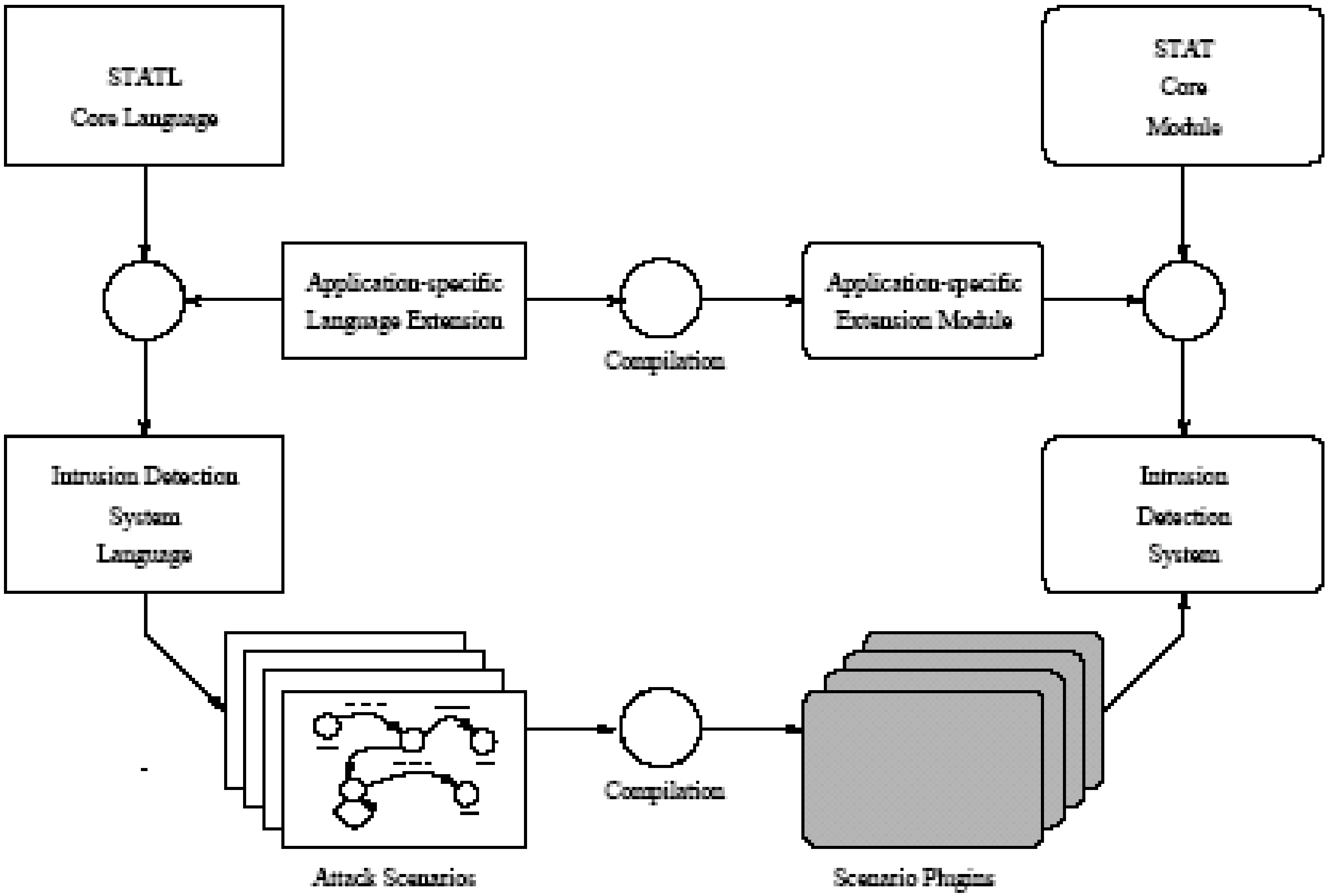
Example Attack Languages – contd.

- UMLSec
  - Some UML elements are extended - use case, class, sequence, state chart, package and deployment diagrams
  - Extension for security are usually based on stereotypes and tag values
- Misuse Cases
  - A use case (scenario) is a sequence of actions which gives service to a user - an actor initiated
  - A misuse case describes an unexpected or unauthorized scenario - a mis-actor initiated
- UMLintr (UML for Intrusion Specifications)
  - A UML profile for intrusion scenarios - notations to specify intrusion scenarios (signatures)

Attack Languages and IDSs

- State Transition Analysis Technique (STAT)
  - Attacks are described as sequences of actions in STATL (STAT Language)
  - An IDS collects information from audit records and network packets
  - The collected information are matched at runtime against a set of attack scenarios expressed in STATL - to determine if an attack has occurred to the system

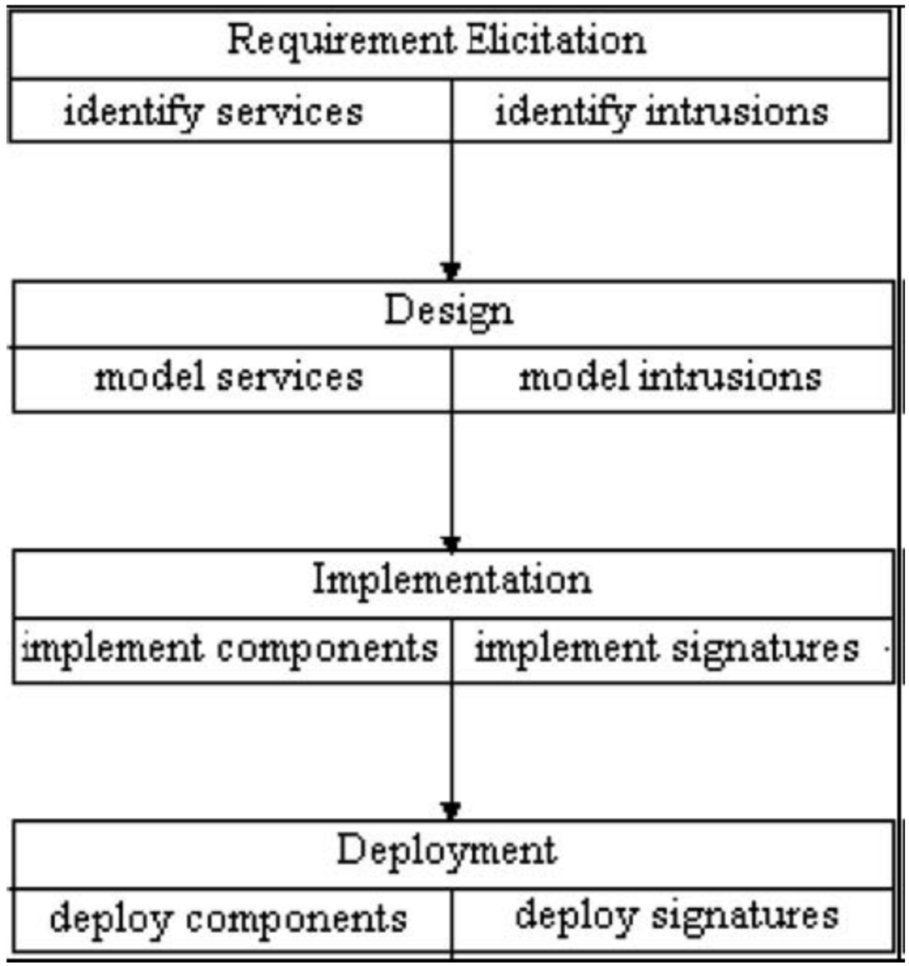
STAT


STATL
Core Language
STAT
Core
Module
Application-specific
Language Extension
Compilation
Application-specific
Extension Module
Intrusion Detection
System
Language
Intrusion
Detection
System
Compilation
Attack Scenarios
Scenario Plugins

STAT – Misuse-Based

- Advantages
  - Can detect co-operative attacks
  - Can detect attacks that span across multiple user sessions
  - Can predict attacks based on current state and may take preventive action
- Disadvantages
  - Attack patterns may not represent too complex attacks
  - Cannot detect if an attack cannot be represented in STATL

Intrusion-Aware Software Development



Software Specification-Based: Misuse-Based

- Advantages
  - Developers do not need to learn a separate language to describe attacks
  - Helps avoid conflicting (e.g., security vs. usability), ambiguous, and redundant requirements
  - Early incorporation of security requirements – it may not be implemented later
- Disadvantage
  - Most software specification languages are not suitable to specify all attacks

Summary

- Security Types
  - Each type emphasize on different aspect of computer security
- Computer Attacks and Defenses
  - A threat is blocked by control (defense) of a vulnerability
  - Attack defense - Intrusion Detection
- Security Engineering
  - Goals, Stakeholders, Life cycle, Sub-disciplines, Related disciplines
  - Security Engineering Process
- Software Security Engineering
  - Software Security
  - Software Risk Management Process
  - Software Specification and Attack Languages

Final Report/Presentation Outline

- Abstract
  - Why was this research carried out?
  - What was done?
  - How was it done?
  - What was found – its implications?
- Keywords (2-6)
- Introduction
  - More details of the above (avoid directly copying from abstract)
  - Paper organization
- Related Work
  - Direct and indirect comparison of your work with other related work
  - Do not just describe other work – compare and contrast
- Main Body
  - What was done?
  - How was it done?
- Experimental Evaluation
  - Evaluation environment
  - What was found - your analysis and explanations

Final Report/Presentation – contd.

- Conclusions and Future Work
  - Similar to abstract but more specific
  - What could not be done - future work to improve your work
- Complete References
- Overall
  - Presentation style - flow of the presentation and clarity: is it understandable? does it progress logically?
- Report Format
  - 8 pages, IEEE CS Proc. paper style (as if you are sending the paper to an IEEE conf.) - borrow everything from previous reports, however, write at least 8 pages!
  - Appendix 2 pages (if needed) – the paper should be understandable independent of the appendix (optional)

Final Report/Presentation – contd.

- Some General Advice
  - The report will be graded independent of your presentation - continue to work on your project after the presentation
  - Use your own examples and figures, if necessary
  - Clearly state your accomplishment & its relationship with other work
  - Use references and/or " " in appropriate places
  - Maintain appropriate ratio for the section lengths
  - Make the report as complete as possible
  - Do not discuss - schedules, personal study, other difficulties – think what you can write on a paper submitted to a conference
- Some other Issues
  - The Final Report is not intended for feedback – only for grading
  - Project work will also be evaluated – source code may be requested!

Lecture Sources

- C. Pfleeger and S. Pfleeger, Security in Computing, Chapter 1 & 3 Prentice-Hall, 2003
- ISO/JTC1/IEC, Information technology -- Security techniques -- Evaluation criteria for IT security -- Part 1, Standard ISO/IEC 15408-(1-3):1999
- C. Landwehr et al., "A Taxonomy of Computer Program Security Flaws," ACM Computing Surveys, vol. 26, no. 3, September 1994.
- R. Anderson, Security Engineering - A Guide to Building Dependable Distributed Systems, Wiley, January 2001
- SSE-CMM: Model Description Document: Version 3.0, Chapter 3.1 $ 3.2, June 2003 (http://www.sse-cmm.org/model/model.asp)
- Northcutt, et al., Inside Network Perimeter Security: The Definitive Guide to Firewalls, Virtual Private Networks (VPNs), Routers, and Intrusion Detection Systems, Chapter 13, Sams, 2002
- J. Viega and G. McGraw, Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley Pub Co, 2001
- Alexander L. Wolf. "Is Security Engineering Really Just Good Software Engineering?, Keynote Talk, ACM , SIGSOFT '04/FSE-12, October 2004, Newport Beach, CA, USA.
- Ivan Flechals, Argela Sasse, and Stephen Hailes, "Bringing Security Home: A process for Developing Secure and Usable Systems," Proc. of the New Security Paradigms Workshop, Ascona, Switzerland, 2003.
- S. Eckmann and et al., STATL: An attack language for state-based intrusion detection, Journal of Computer Security, vol. 10, no. 1/2, pp. 71-104, 2002
- J. Jürjens, Secure Systems Development with UML, Springer-Verlag, December 2003.