IS 2620: Developing Secure Systems

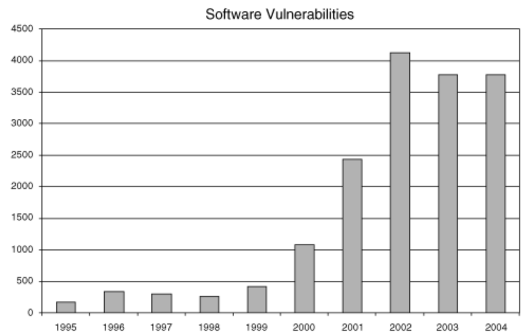
Building Security *In*
Lecture 2

Jan 30, 2007

Software Security

- Renewed interest
  - “idea of engineering software so that it continues to function correctly under malicious attack”
  - Existing software is riddled with design flaws and implementation bugs
  - “any program, no matter how innocuous it seems, can harbor security holes”
- (Check the CBI report)

Software Problem



vulnerabilities
Reported by CERT/CC

- More than half of the vulnerabilities are due to buffer overruns
- Others such as race conditions, design flaws are equally prevalent

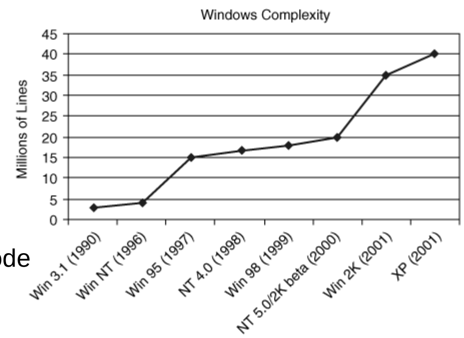
Software security

- It is about
  - Understanding software-induced security risks and how to manage them
  - Leveraging software engineering practice,
  - thinking security early in the software lifecyle
  - Knowing and understanding common problems
  - Designing for security
  - Subjecting all software artifacts to thorough objective risk analyses and testing
- It is a knowledge intensive field

Trinity of trouble

- Three trends
  - Connectivity
    - Inter networked
    - Include SCADA (supervisory control and data acquisition systems)
    - Automated attacks, botnets
  - Extensibility
    - Mobile code – functionality evolves incrementally
    - Web/Os Extensibility
  - Complexity
    - XP is at least 40 M lines of code
    - Add to that use of unsafe languages (C/C++)

Bigger problem today .. And growing

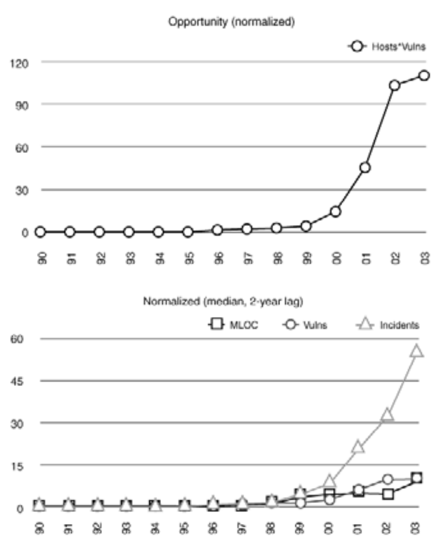


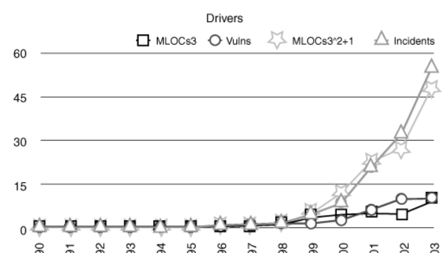
It boils down to …

more code,

more bugs,

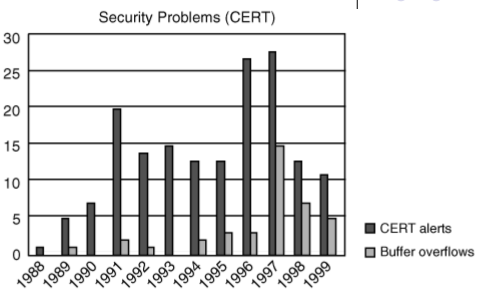
more security problems





Security problems in software

- Defect
  - implementation and design vulnerabilities
  - Can remain dormant
- Bug
  - An implementation level software problem
- Flaw
  - A problem at a deeper level
- Bugs + Flaws
  - leads to Risk



Bug	Flaw
Buffer overflow: stack smashing Buffer overflow: one-stage attacks Buffer overflow: string format attacks Race conditions: TOCTOU Unsafe environment variables Unsafe system calls (fork(), exec(), system()) Incorrect input validation (black list vs. white list	Method over-riding problems (subclass issues) Compartmentalization problems in design Privileged block protection failure (DoPrivilege()) Error-handling problems (fails open) Type safety confusion error Insecure audit log design Broken or illogical access control (role-based access control [RBAC] over tiers) Signing too much code

Solution …
Three pillars of security

Pillar I:
Applied Risk management

- Architectural risk analysis
  - Sometimes called threat modeling or security design analysis
  - Is a best practice and is a touchpoint
- Risk management framework
  - Considers risk analysis and mitigation as a full life cycle activity

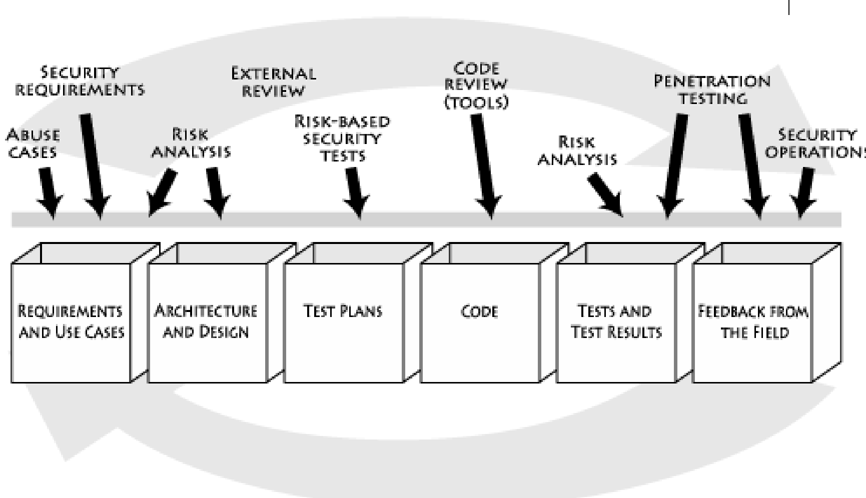
Pillar II:
Software Security Touchpoints

- “Software security is not security software”
  - Software security
    - is system-wide issues (security mechanisms and design security)
    - Emergent property
- Touchpoints in order of effectiveness (based on experience)
  - Code review (bugs)
  - Architectural risk analysis (flaws)
    - These two can be swapped
  - Penetration testing
  - Risk-based security tests
  - Abuse cases
  - Security requirements
  - Security operations

Pillar II: (contd.)

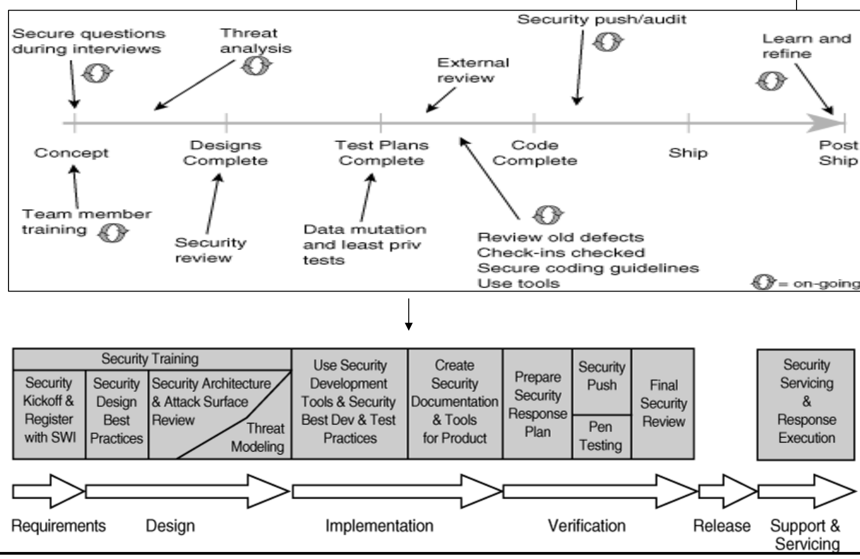
- Many organization
  - Penetration first
    - Is a reactive approach
- CR and ARA can be switched however skipping one solves only half of the problem
- Big organization may adopt these touchpoints simultaneously

Pillar II: (contd.)

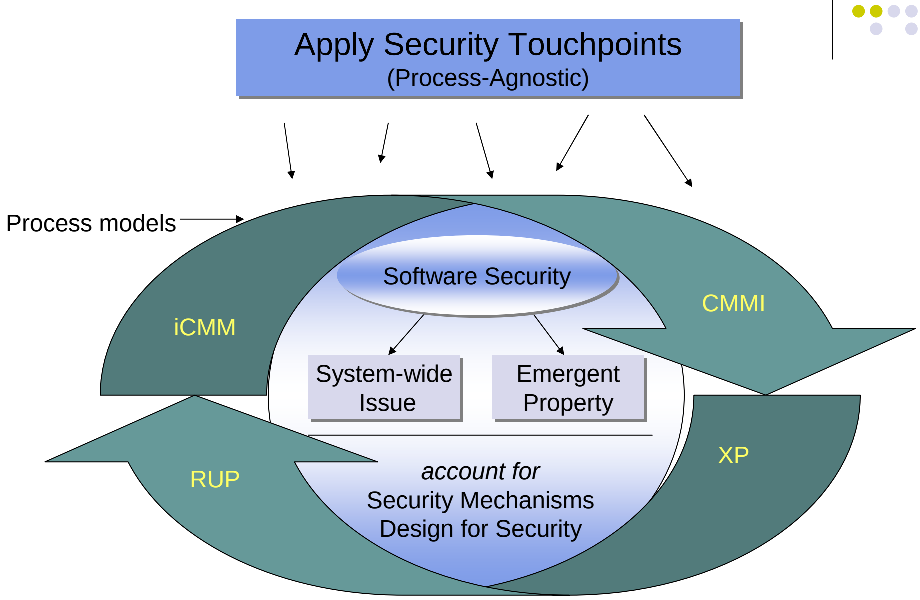


Software security best practices applied to various software artifacts

Pillar II: (contd.) Microsoft's move ..



Pillar II: (contd.)



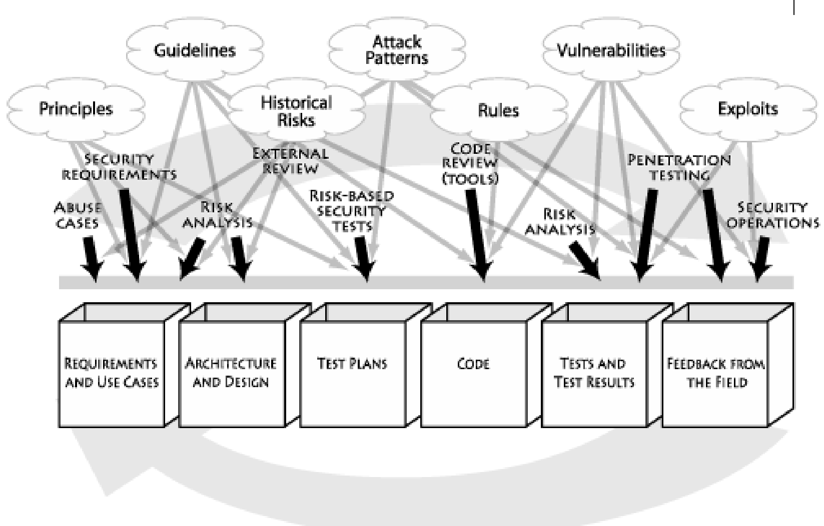
Pillar III: Knowledge

- Involves
  - Gathering, encapsulating, and sharing security knowledge
- Software security knowledge catalogs
  - Principles
  - Guidelines
  - Rules
  - Vulnerabilities
  - Exploits
  - Attack patterns
  - Historical risks

Can be put into three categories

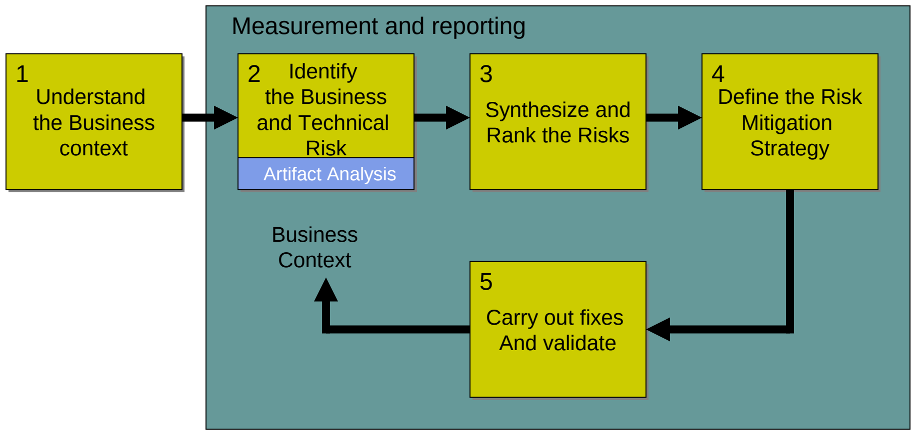
Prescriptive knowledge
Diagnostic knowledge
Historical knowledge

Pillar III: Knowledge catalogs to s/w artifacts



Risk management framework: Five Stages

- RMF occurs in parallel with SDLC activities



Stage 1: Understand Business Context

- Risk management
  - Occurs in a business context
  - Affected by business motivation
- Key activity of an analyst
  - Extract and describe business goals – clearly
    - Increasing revenue; reducing dev cost; meeting SLAs; generating high return on investment (ROI)
  - Set priorities
  - Understand circumstances
- Bottomline – answer the question
  - who cares?

Stage 2: Identify the business & technical risks

- Business risks have impact
  - Direct financial loss; loss of reputation; violation of customer or regulatory requirements; increase in development cost
- Severity of risks
  - Should be capture in financial or project management terms
- Key is –
  - tie technical risks to business context

Stage 3: Synthesize and rank the risks

- Prioritize the risks alongside the business goals
- Assign risks appropriate weights for resolution
- Risk metrics
  - Risk likelihood
  - Risk impact
  - Number of risks mitigated over time

Stage 4: Risk Mitigation Strategy

- Develop a coherent strategy
  - For mitigating risks
  - In cost effective manner; account for
    - Cost Implementation time
    - CompletenessImpact
    - Likelihood of success
- A mitigation strategy should
  - Be developed within the business context
  - Be based on what the organization can afford, integrate and understand
  - Must directly identify validation techniques

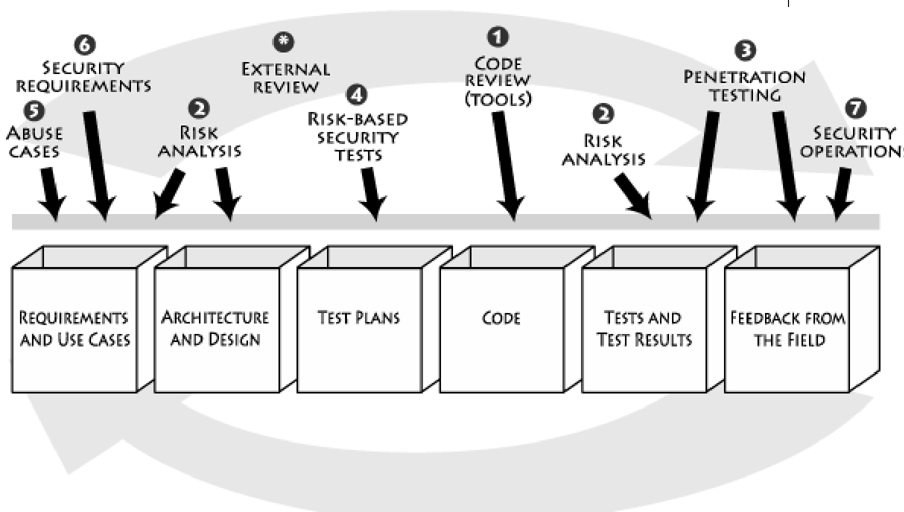
Stage 5: Carry out Fixes and Validate

- Execute the chosen mitigation strategy
    - Rectify the artifacts
  - Measure completeness
  - Estimate
    - Progress, residual risks
- Validate that risks have been mitigated
  - Testing can be used to demonstrate
  - Develop confidence that unacceptable risk does not remain

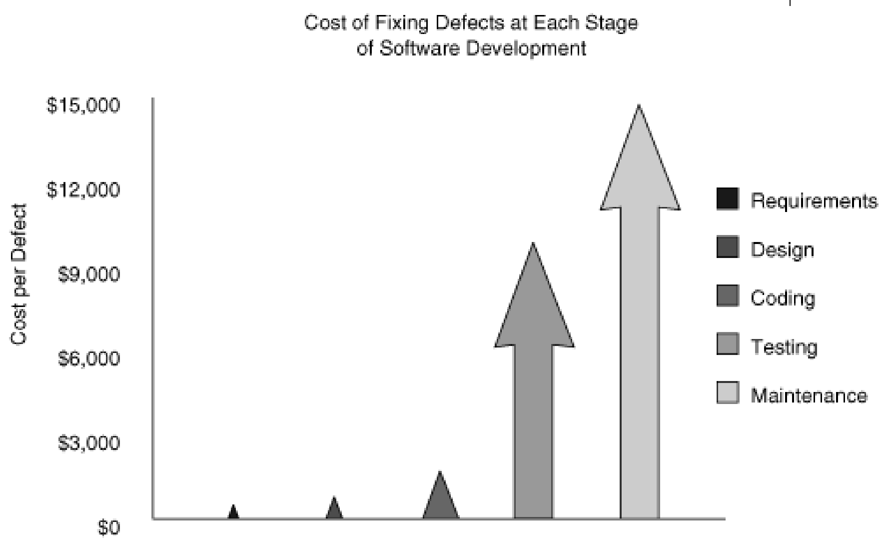
RMF - A Multi-loop

- Risk management is a continuous process
  - Five stages may need to be applied many times
  - Ordering may be interleaved in different ways
    - Risk can emerge at any time in SDLC
      - One way – apply in each phase of SDLC
    - Risk can be found between stages
- Level of application
  - Primary – project level
    - Each stage must capture complete project
  - SDLC phase level
  - Artifact level
- It is important to know that RM is
  - Cumulative
  - At times arbitrary and difficult to predict

Seven Touchpoints



Cost of fixing defect at each stage



Code review

- Focus is on implementation bugs
  - Essentially those that static analysis can find
  - Security bugs are real problems – but architectural flaws are just as big a problem
    - Code review can capture only half of the problems
  - E.g.
    - Buffer overflow bug in a particular line of code
  - Architectural problems are very difficult to find by looking at the code
    - Specially true for today's large software

Code review

- Taxonomy of coding errors
  - Input validation and representation
    - Some source of problems
      - Metacharacters, alternate encodings, numeric representations
      - Forgetting input validation
      - Trusting input too much
      - Example: buffer overflow; integer overflow
  - API abuse
    - API represents contract between caller and callee
    - E.g., failure to enforce principle of least privilege
  - Security features
    - Getting right security features is difficult
    - E.g., insecure randomness, password management, authentication, access control, cryptography, privilege management, etc.

Code review

- Taxonomy of coding errors
  - Time and state
    - Typical race condition issues
    - E.g., TOCTOU; deadlock
  - Error handling
    - Security defects related to error handling are very common
    - Two ways
      - Forget to handle errors or handling them roughly
      - Produce errors that either give out way too much information or so radioactive no one wants to handle them
    - E.g., unchecked error value; empty catch block

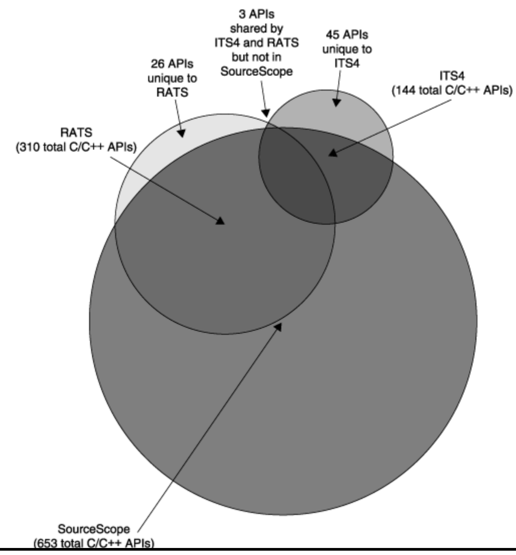
Code review

- Taxonomy of coding errors
  - Code quality
    - Poor code quality leads to unpredictable behavior
    - Poor usability
    - Allows attacker to stress the system in unexpected ways
    - E.g., Double free; memory leak
  - Encapsulation
    - Object oriented approach
    - Include boundaries
    - E.g., comparing classes by name
  - Environment
    - Everything outside of the code but is important for the security of the software
    - E.g., password in configuration file (hardwired)

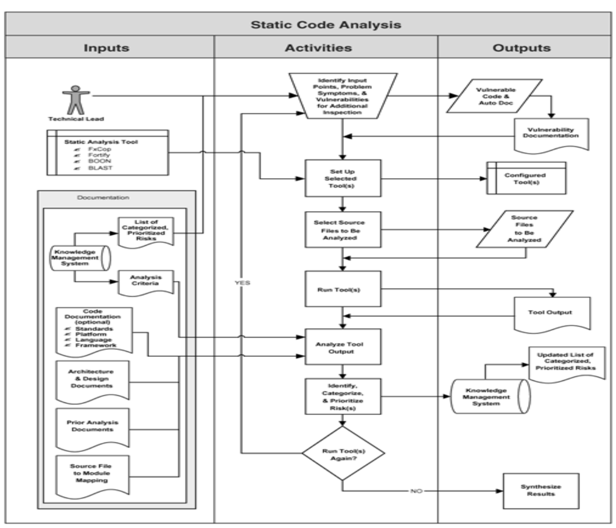
Code review

- Static analysis tools
  - False negative (wrong sense of security)
    - A sound tool does not generate false negatives
  - False positives
  - Some examples
    - ITS4 (It's The Software Stupid Security Scanner);
    - RATS; Flawfinder

Rules overlap



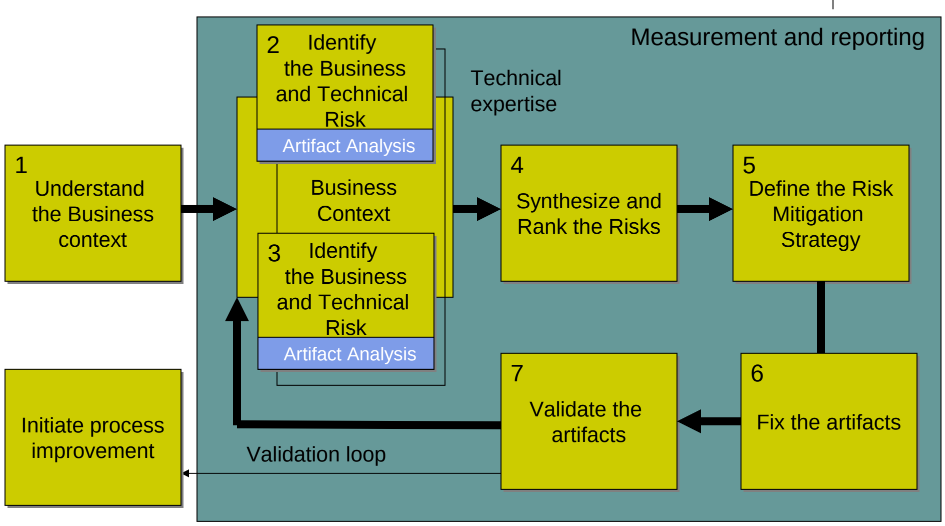
Cigital Static analysis process

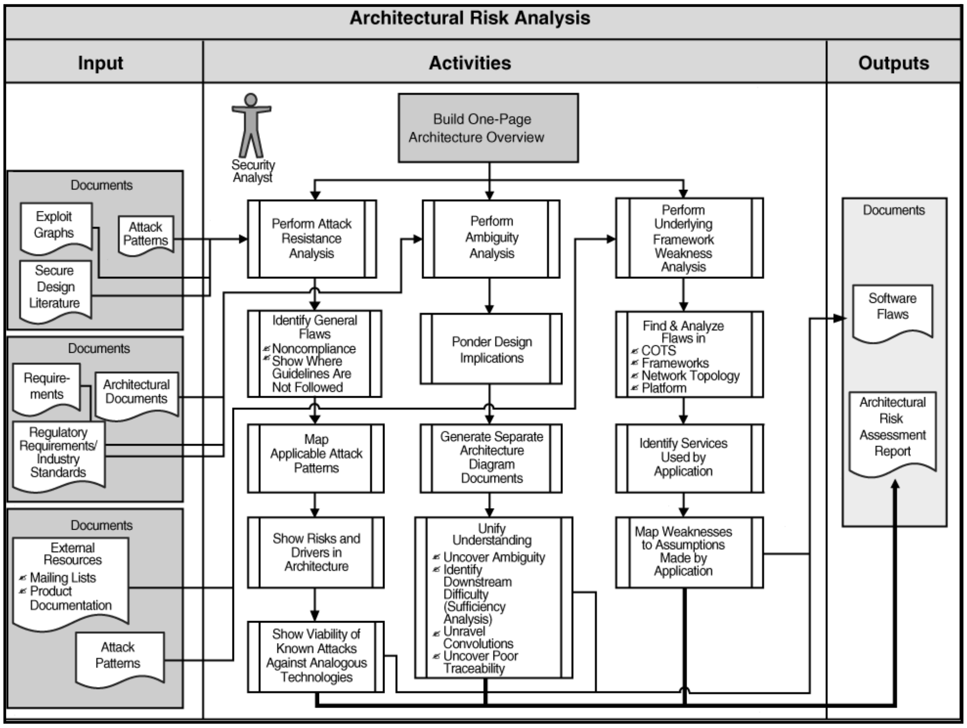


Architectural risk analysis

- Design flaws
  - about 50% of security problem
  - Can't be found by looking at code
    - A higher level of understanding required
- Risk analysis
  - Track risk over time
  - Quantify impact
  - Link system-level concerns to probability and impact measures
  - Fits with the RMF

ARA within RMF





ARA process

- Attack resistance analysis
  - Steps
    - Identify general flaws using secure design literature and checklists
      - Knowledge base of historical risks useful
    - Map attack patterns using either the results of abuse case or a list of attack patterns
    - Identify risk based on checklist
    - Understand and demonstrate the viability of these known attacks
      - Use exploit graph or attack graph

- Note: particularly good for finding known problems

ARA process

- Ambiguity analysis
  - Discover new risks – creativity requried
  - A group of analyst and experience helps – use multiple points of view
    - Unify understanding after independent analysis
  - Uncover ambiguity and inconsistencies
- Weakness analysis
  - Assess the impact of external software dependencies
  - Modern software
    - is built on top of middleware such as .NET and J2EE
    - Use DLLs or common libraries
  - Need to consider
    - COTS
    - Framework
    - Network topology
    - Platform
    - Physical environment
    - Build environment

Software penetration testing

- Most commonly used today
- Currently
  - Outside->in approach
  - Better to do after code review and ARA
  - As part of final preparation acceptance regimen
  - One major limitation
    - Almost always a too-little-too-late attempt at the end of a development cycle
      - Fixing things at this stage
        - May be very expensive
        - Reactive and defensive

Software penetration testing

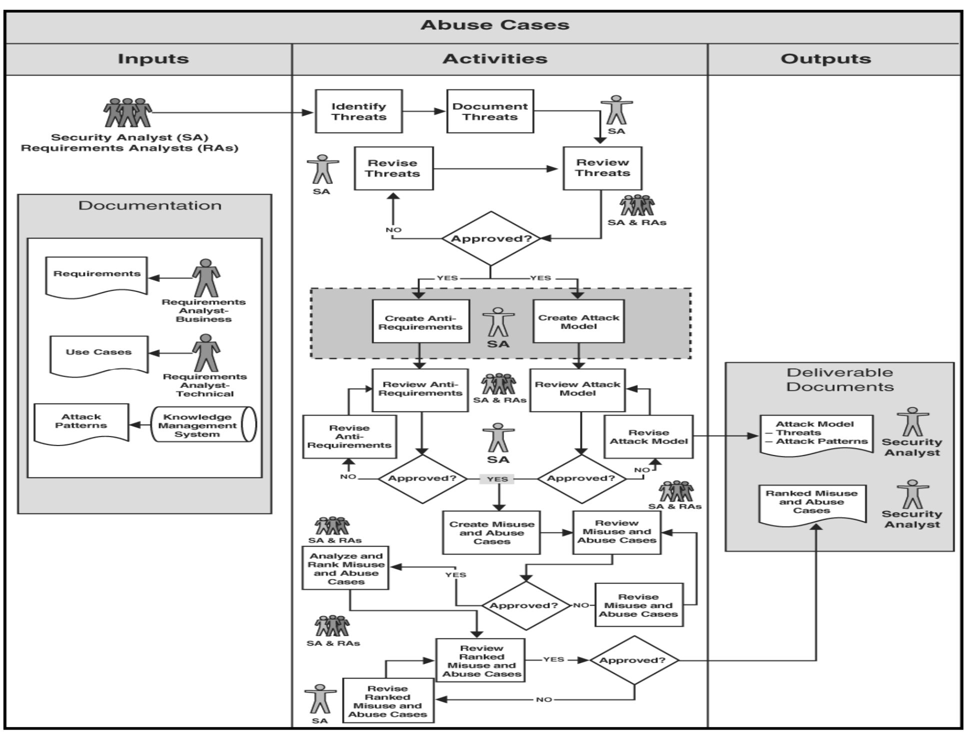
- A better approach
  - Penetration testing from the beginning and throughout the life cycle
  - Penetration test should be driven by perceived risk
  - Best suited for finding configuration problems and other environmental factors
  - Make use of tools
    - Takes care of majority of grunt work
    - Tool output lends itself to metrics
    - Eg.,
      - fault injection tools;
      - attacker's toolkit: disassemblers and decompilers; coverage tools monitors

Risk based security testing

- Testing must be
  - Risk-based
  - grounded in both the system's architectural reality and the attacker's mindset
    - Better than classical black box testing
  - Different from penetration testing
    - Level of approach
    - Timing of testing
      - Penetration testing is primarily on completed software in operating environment; outside->in

Risk based security testing

- Security testing
  - Should start at feature or component/unit level testing
  - Must involve two diverse approaches
    - Functional security testing
      - Testing security mechanisms to ensure that their functionality is properly implemented
    - Adversarial security testing
      - Performing risk-based security testing motivated by understanding and simulating the attacker's approach



Abuse cases

- Creating anti-requirements
  - Important to think about
    - Things that you don't want your software to do
    - Requires: security analysis + requirement analysis
  - Anti-requirements
    - Provide insight into how a malicious user, attacker, thrill seeker, competitor can abuse your system
    - Considered throughout the lifecyle
      - indicate what happens when a required security function is not included

Abuse cases

- Creating an attack model
  - Based on known attacks and attack types
  - Do the following
    - Select attack patterns relevant to your system – build abuse case around the attack patterns
    - Include anyone who can gain access to the system because threats must encompass all potential sources
  - Also need to model attacker

Security requirements and operations

- Security requirements
  - Difficult tasks
  - Should over both overt functional security and emergent characteristics
    - Use requirements engineering approach
- Security operations
  - Integrate security operations
    - E.g., software security should be integrated with network security