Secure Development Processes

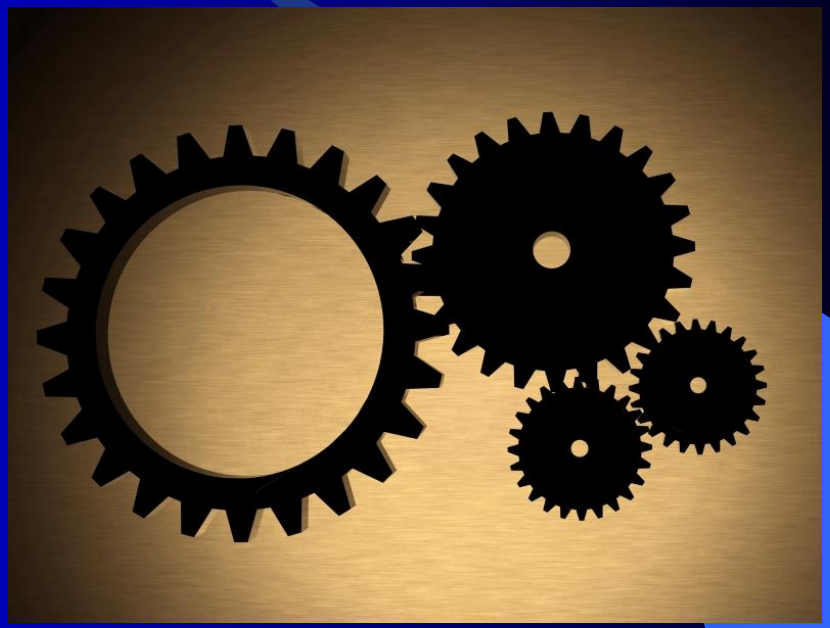
SecAppDev 2008

What's the problem?

- Writing *secure* software is tough
- Newcomers often are overwhelmed
  - Fear of making mistakes can hinder
- Tend to delve into security superficially
  - Pen testing
  - Purchase a source code analyzer
- Business needs software dev to be
  - Predictable
  - Repeatable
  - Reliable
- This can drive the need for a solid process
  - Consistently applied

Consider a Secure SDLC

- Several to choose from
- Enough good in each to consider all
  - Look carefully at each author's perspective
- Apply consistently and measure



Who are the players?

- Microsoft
  - Secure Development Lifecycle
  - "The Security Development Lifecycle," Michael Howard and Steve Lipner, Microsoft Press, ISBN 978-0-7356-2214-2
- Cigital
  - "Touchpoint" process
  - "Software Security: Building Security In," Gary McGraw, Addison-Wesley, ISBN 0-321-35670-5
  - http://BuildSecurityIn.US-CERT.gov
- OWASP
  - Comprehensive Lightweight Application Security Process (CLASP)
  - http://www.owasp.org/index.php/OWASP_CLASP_Project

MS-SDL Overview

- Consists of 12 stages
  - Stage 0: Education and awareness
  - Stage 1: Project inception
  - Stage 2: Define and follow design best practices
  - Stage 3: Product risk assessment
  - Stage 4: Risk analysis
  - Stage 5: Creating security documents, tools, and best practices for customers
  - Stage 6: Secure coding policies

MS-SDL Overview, cont'd

- Stage 7: Secure testing policies
- Stage 8: The security push
- Stage 9: The final security review
- Stage 10: Security response planning
- Stage 11: Product release
- Stage 12: Security response execution

Stage 0: Education and Awareness

- Good stuff, make sure your developers understand what needs to be done **and why**
- Knowledge management should include
  - Attacks and how to prevent, detect, respond
  - Language pitfalls
  - Secure design patterns
  - How to apply the SDLC
- Developers should get annual training
  - Novice through expert

Stage 1: Project Inception

- Decide on each of the following:
  - Should app be written to SDL?
  - Security advisor
  - Security leadership team
    - Roles, responsibilities, expectations
  - Bug tracking process
  - “Bug bar”

Stage 2: Design Best Practices

- Define and follow, based on
  - Secure design principles
    - Think Saltzer and Schroeder
  - Attack surface analysis and reduction

Stage 3: Product Risk Assessment

- Analyze the product's functions and their "danger" levels
  - Use their sample questionnaire as a starting point
- Determine the privacy impact
- How much effort should be applied?

Stage 4: Risk Analysis

- This one really comes down to
  - Threat modeling
  - Using threat model to aid code review
  - Using threat model to aid testing
  - Determine key success factors and metrics
- Guided by
  - STRIDE (Spoofing, Tampering, Repudiation, Info disclosure, DoS, Elevation)
  - DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability)

Stage 5: Customer focus

- Creating security documents, tools, and best practices for customers
  - Help your customers run your application securely
  - Security features, settings, file access controls, etc.

Stage 6: Secure Coding Policies

- Ensure each of the following
  - Use latest compiler, library, and features
  - Do source code analysis (with tools)
  - Avoid banned functions (and don't re-invent them)
  - Avoid exploitable constructs or designs
  - Follow a secure coding checklist

Stage 7: Secure Testing Policies

- Basically, get (way) beyond the penetration test
  - Fuzzing
  - Penetration testing
  - Run-time verification
  - Update threat models
  - Update attack surface

Stage 8: The Security Push

- Basically, a concerted effort to ensure everything was done right, just before launch
  - Check and double check everything

Stage 9: Final Security Review

- Fundamentally, answer whether the product is ready to ship
  - Validate unfixed bugs (and why)
  - Verify we did all that other stuff
  - Team sign-off

Stage 10: Security Response Planning

- What do we do when things go wrong?
  - Specifically, the *dev* team
  - Plan for it
  - Designate the team
  - Ensure facilities are available

Stage 11: Product Release

- Does it dump core? Ship it!
- Final coordination of product security issues
  - Product support staff ready?
  - Update server functional?

Stage 12: Security Response Execution

- Follow the plan
  - Don’t (kernel) panic
- Iterate as necessary
- Capture lessons learned
- Feedback loop to product dev team

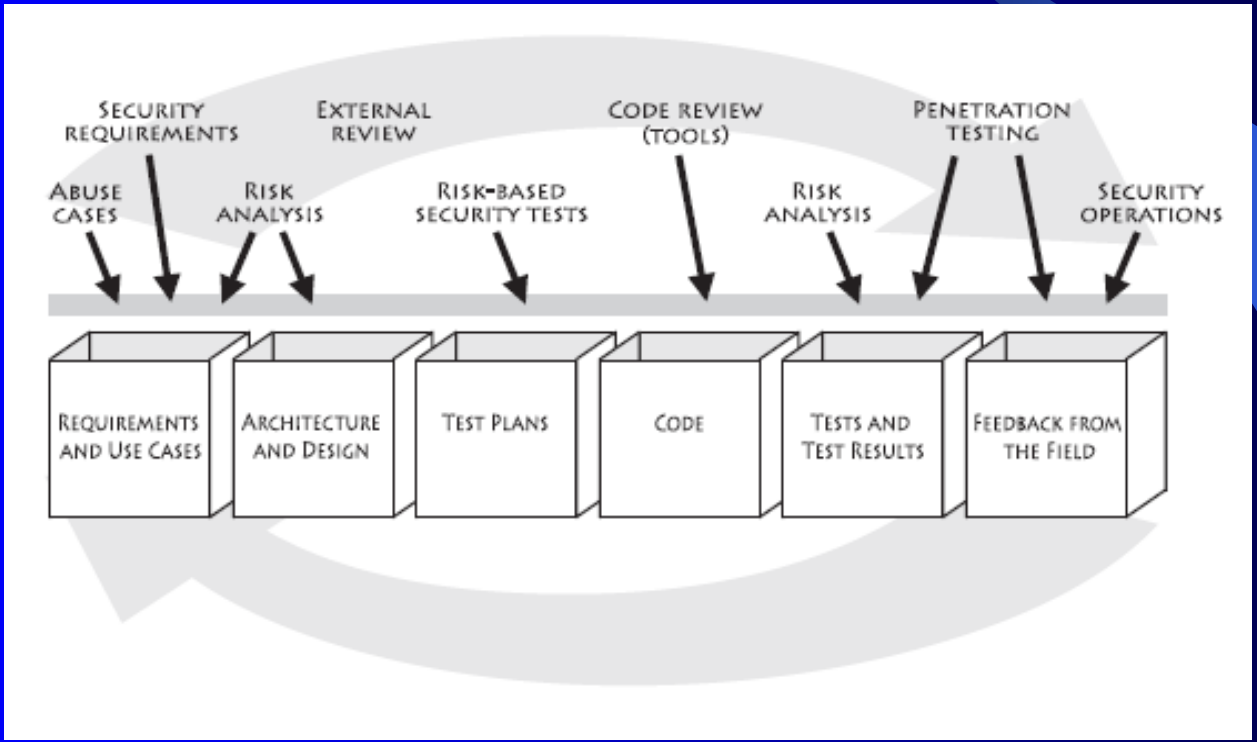
Cigital's "Touchpoints"

- Built by McGraw et al over time
  - Perspective is consulting services
- Consists of three pillars
  - Risk management
  - Knowledge
  - Touchpoints

Artifact-driven

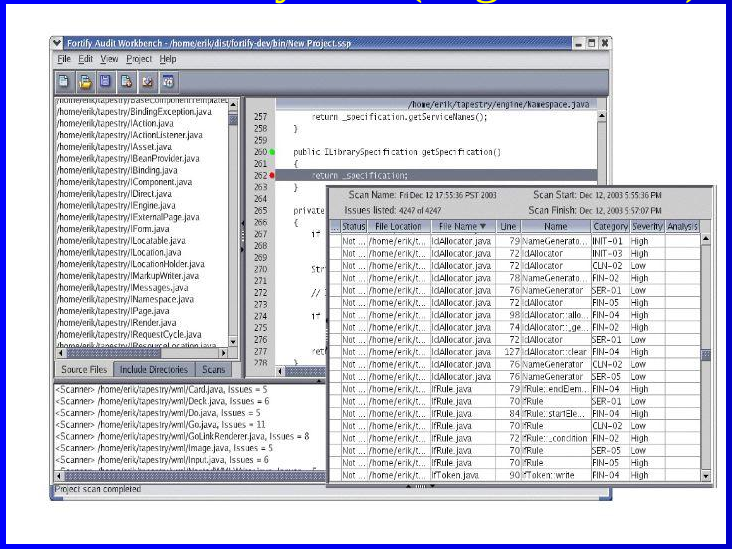
- Touchpoints represent process-agnostic reviews that can be done on each dev artifact
  - Enables the security effort to adapt to any SDLC methodology
- Guiding principle is to not change dev process, but to deeply integrate with it

The Touchpoints



Touchpoint 1: Code review

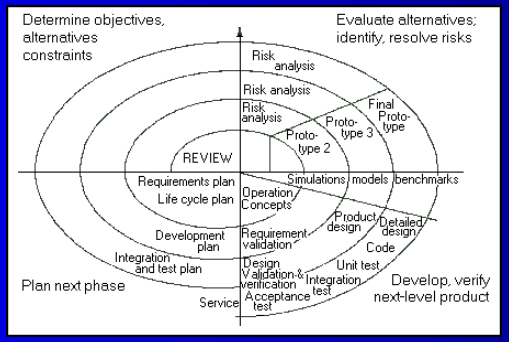
- Code review is a necessary evil
- Better coding practices make the job easier
- Automated tools help catch silly errors
  - Fortify/dev (Cigital rules)

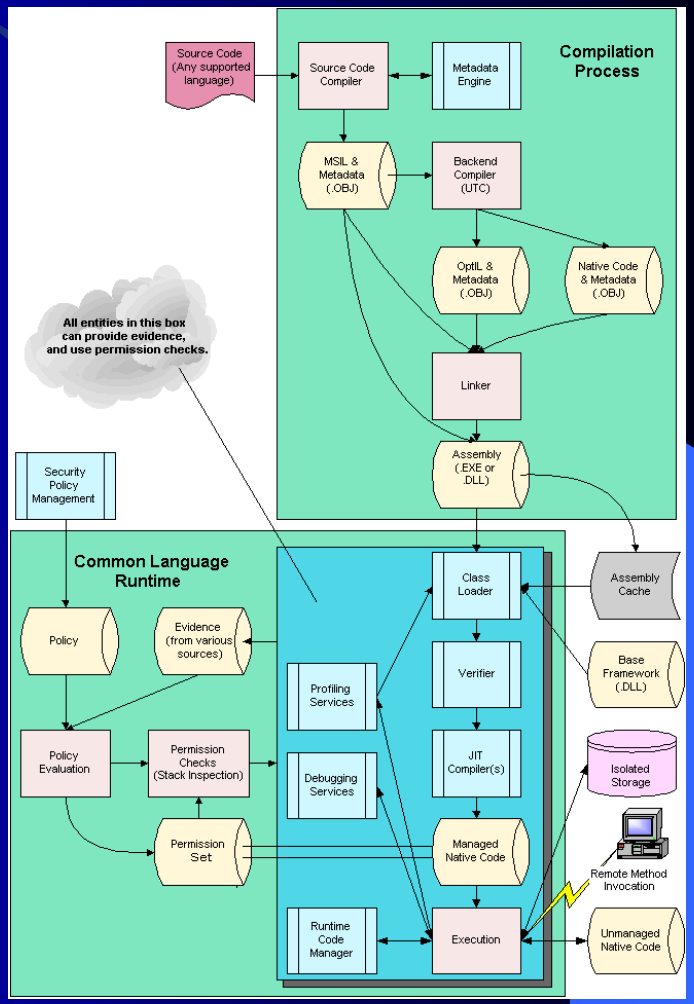


- Implementation errors do matter
  - Buffer overflows can be uncovered with static analysis
  - Fortify SCA
    - Over 500 C/C++ rules
    - Over 100 Java rules
- Tracing back from vulnerable location to input is critical
  - Software exploits
  - Attacking code

Touchpoint 2: Architectural risk analysis

- Build a one page white board design model
- Use hypothesis testing to categorize risks
  - Threat modeling/Attack patterns
- Rank risks
- Tie to business context
- Suggest fixes
- Repeat





Touchpoint 3: Penetration testing

- A very good idea since software is bound in an environment
- How does the complete system work in practice?
  - Interaction with network security mechanisms
  - Firewalls
  - Applied cryptography
- Penetration testing should be driven by risks uncovered throughout the lifecycle
- Not a silver bullet!

Touchpoint 4: Security testing

- Test security functionality
  - Cover non-functional requirements
  - Security software probing
- Risk-based testing
  - Use architectural risk analysis results to drive scenario-based testing
  - Concentrate on what “you can’t do”
  - Think like an attacker
  - Informed red teaming

Touchpoint 5: Abuse cases

- Use cases formalize normative behavior (and assume correct usage)
- Describing non-normative behavior is a good idea
  - Prepare for abnormal behavior (attack)
  - Misuse or abuse cases do this
  - Uncover exceptional cases
- Leverage the fact that designers know more about their system than potential attackers do
- Document explicitly what the software will do in the face of illegitimate use

- Think like an attacker!

Touchpoint 6: Security requirements

- Some security functionality maps naturally to clear requirements
  - Medical data should be cryptographically protected
  - Strongly authenticate users
  - Meet GLBA regulatory guidelines
- But do not forget that security is an emergent property of a complete system
  - An attacker needs to find only one hole
  - “Do not allow buffer overflows” is not much of a requirement!
  - “Make it secure” is vague

Touchpoint 7: Security operations

- Use your resources!
- Network security people know an awful lot about real attacks
- Involve knowledgeable security people in as many touchpoint activities as possible
- Fine tune the deployed environment to the specific needs of your application
  - "Standard OS build" process is not enough

OWASP's CLASP

- Built on seven best practices
  - Institute awareness programs
  - Perform application assessments
  - Capture security requirements
  - Implement secure dev processes
  - Build vulnerability remediation procedures
  - Define and monitor metrics
  - Publish operational security guidelines

OWASP's CLASP

- Built on seven best practices
  - Institute awareness programs
  - Perform application assessments
  - Capture security requirements
  - Implement secure dev processes
  - Build vulnerability remediation procedures
  - Define and monitor metrics
  - Publish operational security guidelines

Documentation

- CLASP is open source and available for download:
  - http://www.list.org/~chandra/clasp/OWASP-CLASP.zip

The Good

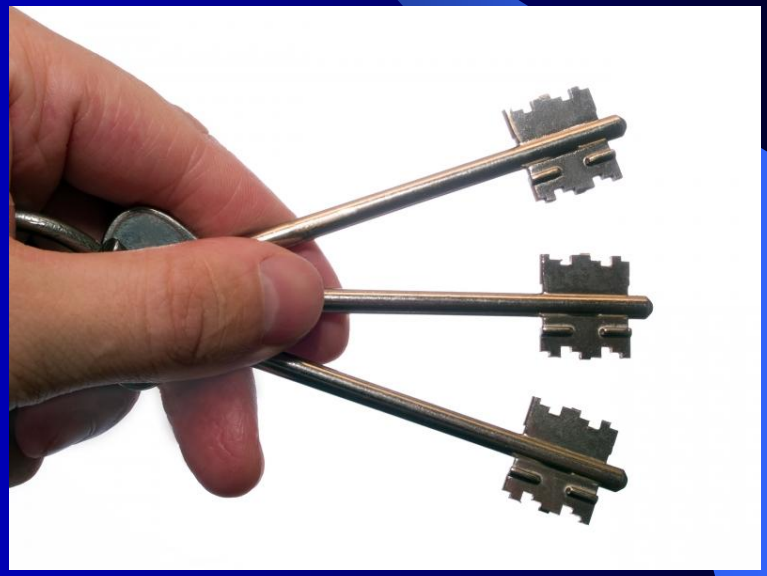
- Microsoft
  - Roles and responsibilities
  - Planning for incidents
  - Customer tips
  - Positive practices
  - Testing
- Cigital
  - Review-based
  - Depth of ARA
  - Code reviews
- OWASP
  - Free and open
  - Security requirements
  - Metrics

The Not-So-Good

- Microsoft
  - Pretty heavy
  - Designed for MS
- Cigital
  - Review-centric
  - Light on positive practices
- OWASP
  - Lots of details yet to be finished

Considerations in Choosing

- One size does NOT fit all
- Cultural issues
  - Dev org size
  - How "process heavy" are you now?
  - Across entire organization



Plan Your Own Hybrid

- Look at each process
- Which components are likely to work best *for you*?
  - Feasibility is vital
  - Sometimes *best* isn't better
- Think things through carefully

Plan of Action

- What is in place now?
- Target process
- Gap analysis
- Chart a course
  - Small steps
  - Defect data helps to prioritize steps
- Buy-in is essential

Other Considerations

- Designate a lead
  - Be available to answer questions
- Document your process
- Provide clear guidelines on how to implement
- Some developers "allergic" to process
- Allow for feedback
  - Adapt as necessary
- Publish results
  - Tips and pitfalls
  - Case studies
- Applying consistently is important
- None of this will happen by itself

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com
http://www.KRvW.com