Software Security: State of the Practice

Diana Kelley
Partner
SecurityCurve

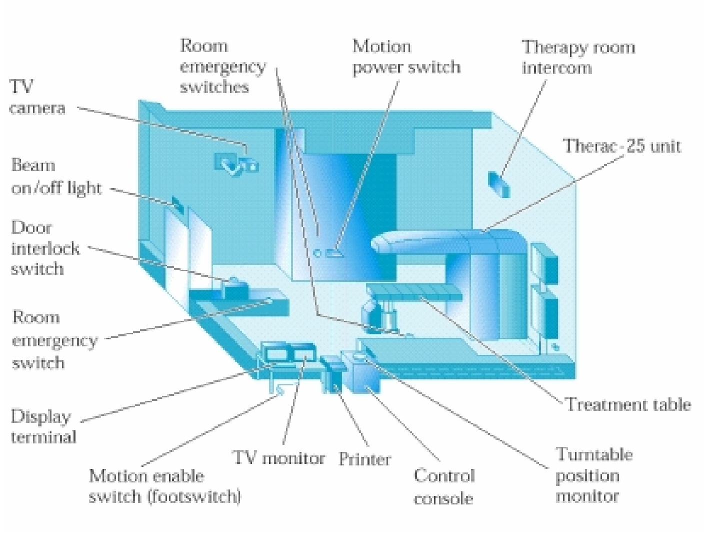
Agenda

- Why Software Security Matters
- Vulnerabilities and Risk in Software
- Building Security In
- Making it Happen

What do These have in Common?

2005 Toyota Prius

Therac-25 Radiation Therapy Machine



Miele G885 SC Dishwasher

Software that Failed

- The dishwasher . . . was rendered useless after a power outage. Its software got knocked out."
  http://www.baselinemag.com/print_article/0,3668,a=35839,00.asp

- "Prius hybrids dogged by software. . . stall or shut down at highway speeds"
  http://money.cnn.com/2005/05/16/Autos/prius_computer/index.htm?cnn=yes

- Six known accidents involved massive overdoses by the Therac-25 -- with resultant deaths and serious injuries."
  http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html

Chemical Bank ATM Incident

- "…a ***single line in an updated computer program*** . . . caused the bank to process every withdrawal and transfer at its automated teller machines twice. Thus a person who took ***$100 from a cash machine had $200 deducted***, although the receipt only indicated a withdrawal of $100."

http://query.nytimes.com/gst/fullpage.html?res=9B00E7D7173BF93BA25751C0A962958260

Royal Bank of Canada Error

- "After a ***software upgrade went badly*** awry last week, the holders of some ***10 million accounts*** at the bank had to ***wait days in some cases for deposits to be credited*** or prearranged payments to be completed."
- …"the problems started with ***a routine programming update*** by the information technology staff. . . . the new software was written in-house"

http://query.nytimes.com/gst/fullpage.html?res=9A06E4D81131F934A35755C0A9629C8B63

RBC Royal Bank

Software Reliability

- **Requires that proper processes and procedures are following during development**
  - Means “building security in” from the beginning
- **Benefits**
  - More reliable software
  - Fewer “gotchas” in pre-production testing
  - Or, worst-case, in deployment
  - If done right – less expensive software/development costs

The OWASP Top Ten

- A1 - Cross Site Scripting (XSS)
- A2 - Injection Flaws
- A3 - Malicious File Execution
- A4 - Insecure Direct Object Reference
- A5 - Cross Site Request Forgery (CSRF) A CSRF
- A6 - Information Leakage and Improper Error Handling
- A7 - Broken Authentication and Session Management
- A8 - Insecure Cryptographic Storage
- A9 - Insecure Communications
- A10 - Failure to Restrict URL Access

Quick Example – SQL Injection

- **Ability to show orders from a table in a SQL DB**
  - Correct Usage
    - User enters in Name field = Kenny
    - Result

```
  SELECT * FROM OrdersTable WHERE CustomerName =
‘Kenny'
```

  - Exploit Usage
    - Attacker enters Name and SQL Command
      - `Kenny;drop table OrdersTable--‘`
      - Semi colon triggers end of query begins a new one
    - Result

```
SELECT * FROM OrdersTable WHERE CustomerName =
  ‘Kenny';drop table OrdersTable--‘
```

  - What happens to the Orders table?

SQL Injection in the News

- **April 2008 - nihaorr1**
  - Infected upwards of 100,000 web pages (per the Register)
    - 500,000 per Slashdot
  - Used SQL injection to infect databases
  - Legitimate users (at legitimate but infected sites) were redirected to the attacker site
  - And infected by drive-by malware/Trojan if vulnerable

Why Tools Can't Catch it All

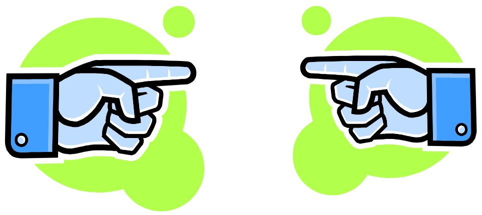
- **Some attacks are not dependent on software failure**
  - Credential Theft
    - Login is valid
    - Activity is approved for that user/role
  - Denial of Service
    - Overloading the application with requests
  - Man in the Middle Attacks
    - Intercept communications
    - Theft – cookies or credentials
    - Inject data into the stream
    - Redirection via bogus DNS

Why Tools Can’t Catch it All

- **Business logic flaws**
  - Abusing a process or function
- **Self-service password recovery**
  - Weak KBA
- **Password lockout**
  - DOS for other users
- **Business logic flaw white paper by Jeremiah Grossman**

http://www.whitehatsec.com/home/assets/WP_bizlogic092407.pdf

Building Security in



- **Check assumptions**
  - And leave finger pointing at the door
- **Is a team effort**
  - "It Takes a Village"
- **Is not the same thing as creating "perfect" code**
  - Unbreakable?
    - Not likely

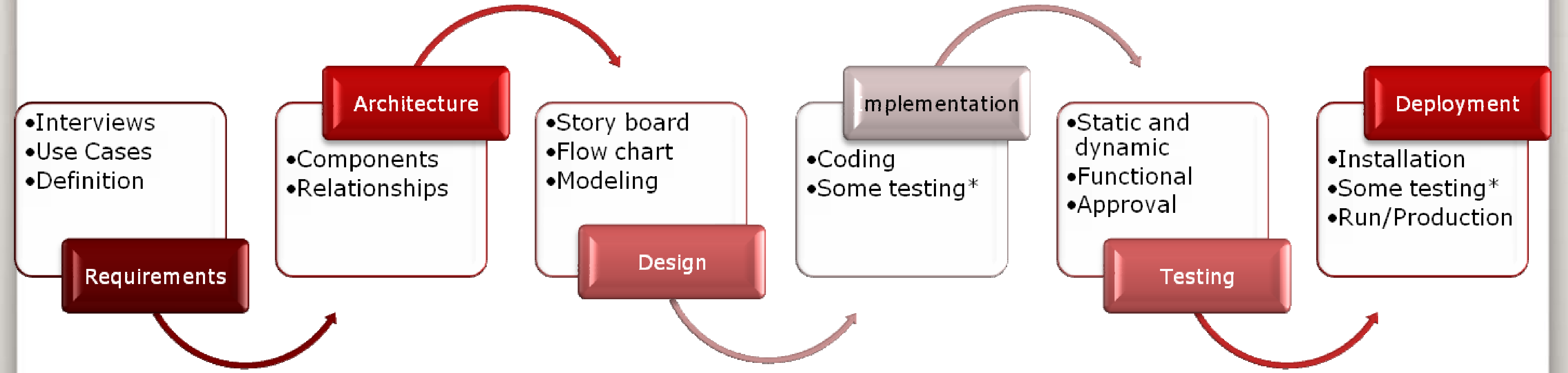
- **Risk assessment**
  - Balancing the risks and consequences
  - Building software that meets the defined risk level

Software Assurance

"Software assurance has as its goal the ability to provide . . . **justifiable confidence that software will consistently exhibit its required properties. . . .** ***security is what*** **enables the software to exhibit those properties even when the software comes under attack."**

From the Information Assurance Technology Analysis Center (IATAC) SOAR on Software Security Assurance
http://iac.dtic.mil/iatac/download/security.pdf

Software Development Lifecycle



Building Security Into the Lifecycle

- Misuse cases
- Security requirements

Requirements

Security Requirements

- **Confidentiality**
  - In use, transit and at rest
  - Mis-use case shoulder surfing a cleartext displayed password
  - Requirement: Mask passwords when typed
- **Integrity**
  - Tamper proofing and tamper evident
  - Mis-use case modification of stored data
  - Requirement: Hash stored data
- **Availability**
  - Ensuring service is available to agreement levels
  - Use case patching or updating the system
  - Requirement: Ability to update without reboot
- **Accountability**
  - Log and verify interaction with the system
  - Mis-use case attacker steals credentials
  - Requirement: Strong authentication

Building Security Into the Lifecycle

Architecture

- Risk assessment
- “Threat modeling”

Building Security Into the Lifecycle

- Risk assessment
- Security test plans

Design

Building Security Into the Lifecycle

Implementation

- Code reviews
- Static code risk testing

Building Security Into the Lifecycle

- Static security testing
- Penetration and dynamic testing
- Risk analysis

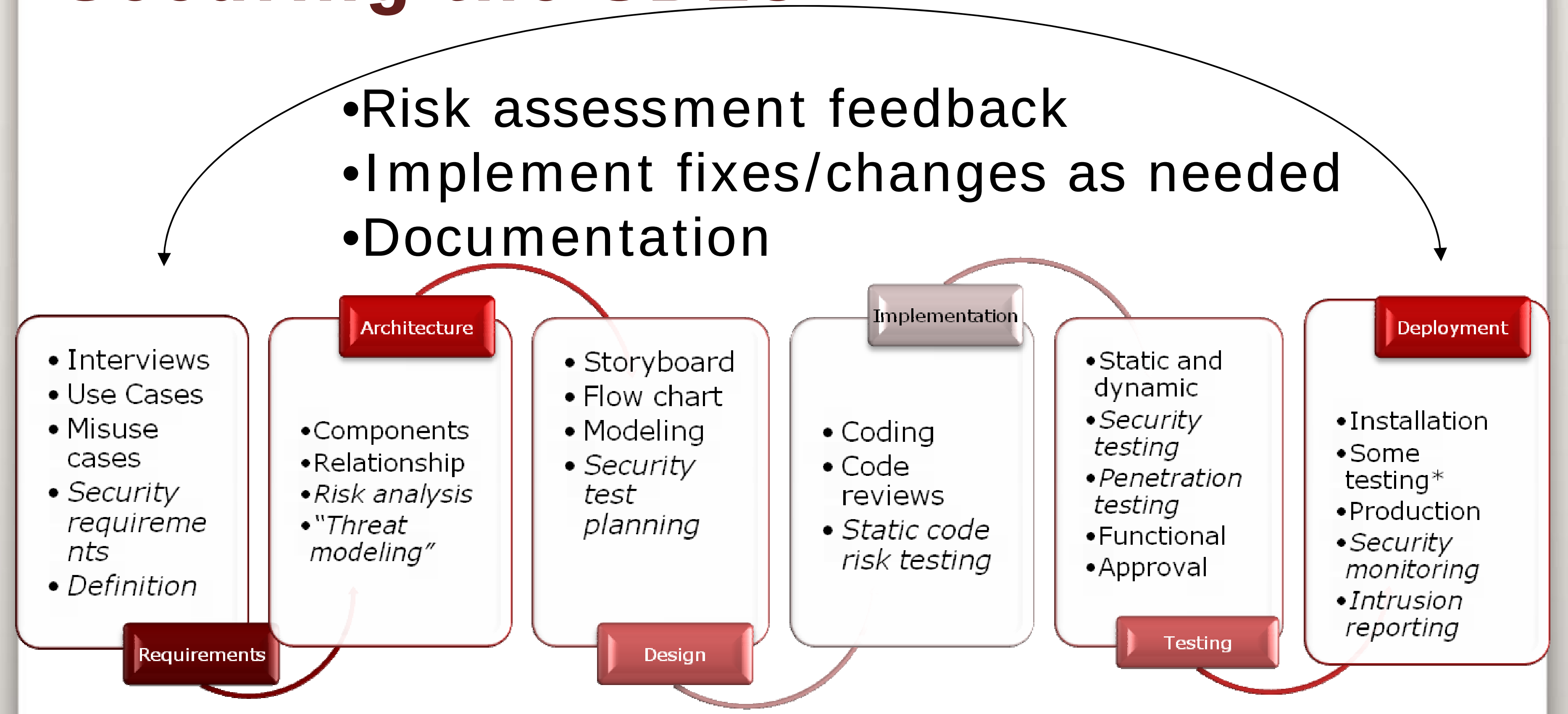
Testing

Building Security Into the Lifecycle

Deployment

- Security monitoring
- Intrusion reporting

Securing the SDLC



Consider the Development Model

- **Less agile**
  - Waterfall
  - Modified Waterfall
- **More agile**
  - Spiral
  - eXtreme Programming (XP)
- **Is one inherently more secure?**
  - Not necessarily
  - Some agility allows for mistakes to be caught and corrected without a full “re-boot”
  - Too much agility trades requirements and design time for speed to code

Additional Secure SDLC Resources

- **Comprehensive, Lightweight Application Security Process, (CLASP)**
  - The Open Web Application Security Project (OWASP)

http://www.owasp.org/index.php/OWASP_CLASP_Project

- **The OWASP Top Ten**
  - And Testing Guide

http://www.owasp.org/index.php/OWASP_Top_Ten_Project
http://www.owasp.org/index.php/Category:OWASP_Testing_Project

- **Cigital's TouchPoints**
  - Software Security: Building Security In by Gary McGraw

http://www.cigital.com/training/touchpoints/

Additional Secure SDLC Resources

- **DHS – Build Security In**
  https://buildsecurityin.us-cert.gov/
  - Top Ten Security Coding Practice
    https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices
- **SAMATE - Software Assurance Metrics And Tool Evaluation**
  https://samate.nist.gov/index.php/Main_Page
- **Microsoft's Security Development Lifecycle (SDL)**
  - A Look Inside the Security Development Lifecycle at Microsoft
    http://msdn.microsoft.com/msdnmag/issues/05/11/SDL/
  - The Security Development Lifecycle by Michael Howard and Steve Lipner

Additional Considerations

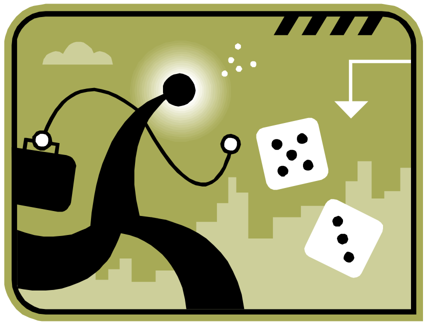
- **Have change control procedures**
  - Impact statements, signoff for changes, and backout procedures
- **Have a process for identifying new vulnerabilities**
- **Test production changes**
  - And educate testers!
- **Interactions with other services**
- **Have separate personnel and environments for production and test**
  - Mask out sensitive data when testing
- **Code review or application firewall consider:**
  - Time constraints
  - Code availability
  - Administrative overhead of firewall configuration

Tools

- **Static source code analysis**
  - Requires access to source code
  - Can be accomplished before build
  - Manual or
  - Automated
    - For developers (inside the IDE)
    - For auditors/testers (as stand alone)
- **Dynamic**
  - Source code not required
  - Tests the product from the view of the “outsider”
  - Best in conjunction with
    - Skilled testers who can tune the products
    - Manual penetration testing to validate tool findings

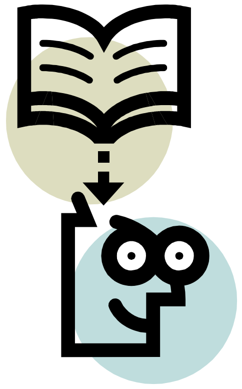
Education

- **For business owners**
  - Risks associated with poorly designed software
  - Value to the final product
- **For assessors and auditors**
  - Common secure coding errors
  - Consequence evaluation
  - Dependencies on key regulations/compliance mandates



Education

- **For testers**
  - Creating misuse and abuse cases
  - Penetration (manual and assisted)
- **For developers**
  - How to write secure code
  - Common coding errors
  - Language specific security training
    - .NET is different from Java
    - Web apps are different from C/C++ apps



Educational Resources

- **For Developers**
  - Certification
  - GIAC Secure Software Programmer (GSSP)
    - Currently Java/JavaEE and C
    - C++, .NET/ASP, PHP, PERL, and others coming soon*

http://www.sans-ssi.org/#cert

- **For Security Professionals**
  - Certified Secure Software Lifecycle Professional- CSSLP^{CM}
  - Offered by ISC2
  - Exams starting in June 2009 – Experience Assessment Now

https://www.isc2.org/cgi-bin/content.cgi?category=1690

Educational Resources

- **Universities – examples:**
  - Carnegie Mellon University (CMU) and University of Ontario (Canada): Secure Software Systems
  - Northeastern University: Engineering Secure Software Systems
  - University of California at Berkeley, Walden University (online): Secure Software Development
  - University of Oxford (UK): Design for Security
- **Commercial Providers**
  - Cigital
  - Neohapsis
  - SecurityInnovation

Making it Happen - Executives

- **Usually focused on**
  - Cost control and ROI
  - Compliance/Regulations
    - “Orange isn’t my color”
  - Metrics and provable results
- **Cost control and ROI**
  - Emphasize improvements to the process and potential cost savings
  - “Software Errors Cost U.S. Economy *$59.5 Billion* Annually” – NIST

http://www.nist.gov/public_affairs/releases/n02-10.htm

Making it Happen - Executives

- **Cost control and ROI**
  - *100 time more expensive* to find and fix problems earlier
  - 40-50% of effort is *avoidable* rework
  - *90%* of downtime comes from (at most) *10%* of defects

Source:Software Defect Reduction Top-10 List, Barry Boehm, USC and Victor Basili, U. of Maryland, Center for Empirically-Based Software Engineering (CeBASE)
http://www.cebase.org/www/AboutCebase/News/top-10-defects.html

Making it Happen - Executives

- **Compliance/regulations**
- **Tools and outsourcing to accomplish goals/meet needs**
- **Metrics and provable results**
  - Dashboards from vendor testing tools
  - Internally gathered metrics
    - Define "success"
      - Reduce severe vulnerabilities
      - Fix vulnerabilities faster
      - Product with fewer vulns in production
  - NIST Software Assurance Metrics And Tool Evaluation (SAMATE)

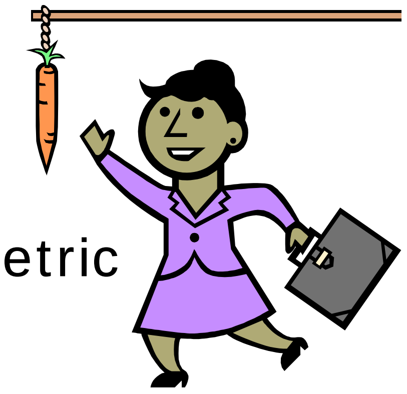
http://samate.nist.gov/index.php/Main_Page

Making it Happen - Partners and Suppliers

- **Usually focused on**
  - Cost control and ROI
  - Making the sale
  - Liability
- **Some controls**
  - Requiring documented use of security in the SDLC
  - Writing SLAs that make payment contingent on meeting them
    - Using tools to measure software security assurance before acceptance
- **Liability**
  - Get legal involved
    - Remuneration if loss occurs
    - Tight SLAs
      - Based on losses and failures
  - But keep in mind
    - Accountability is not transferrable

Making it Happen - Developers

- **Usually focused on**
  - Doing a good job as defined by
    - The company.
    - Their own internal compass.
- **What are the developers in your company rewarded for?**
  - Is it
    - Lines of code?
    - Speed of completion?
    - Match to functions?
  - Make writing low-defect code a success metric
  - Create incentives for teams that build
    - Robust software.
    - That meets corporate software assurance levels.



Making it Happen - Developers

- **A good job – Compass**
  - Provide training to developers
  - Provide tools that will empower
    - Self-checks
    - Learn what works
    - Static source analysis plug-ins for IDEs

Final Thoughts

- **Follow a robust SDLC methodology**
- **Implement risk management at all phases**
  - Bring risk to the table early
  - It'll save money in the long run
  - Define security requirements before implementation
  - Test applications for mis-use cases before production
- **Tools are useful – but not a panacea**
  - Can't fix broken requirements definition
  - Can't scan for business logic errors
- **Education is critical**
  - For all stakeholders