

A Practical Guide to Securing the SDLC

Ryan Berg

*Sr. Architect Security
Research, Rational*

ryan.berg@us.ibm.com

ASC-1340A

Innovate2010

The Rational Software Conference

Let's **build** a smarter planet.

The premiere software and product delivery event.

June 6-10 Orlando, Florida



Why are we here ?

- I had some time to fill and this topic looked interesting
- Application security is the latest *<insert newest internet related buzzword here>*
- Heard there was free food
- I thought this was a different session but am too embarrassed to leave
- The “monster at the end of the book”



Application security is more visible...



Combating Data Theft



Securing Outsourcing



Supporting Compliance



Securing the SDLC

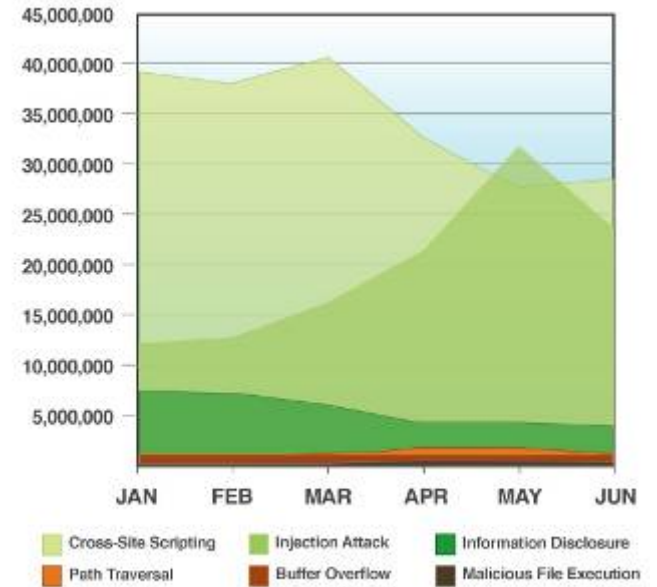
Managers & Developers Are Being Asked Difficult Questions

- What regulations and standards are required?
 - ▶ PCI, HIPAA, FISMA
- What confidential data is at risk?
- What risk threshold is tolerable?

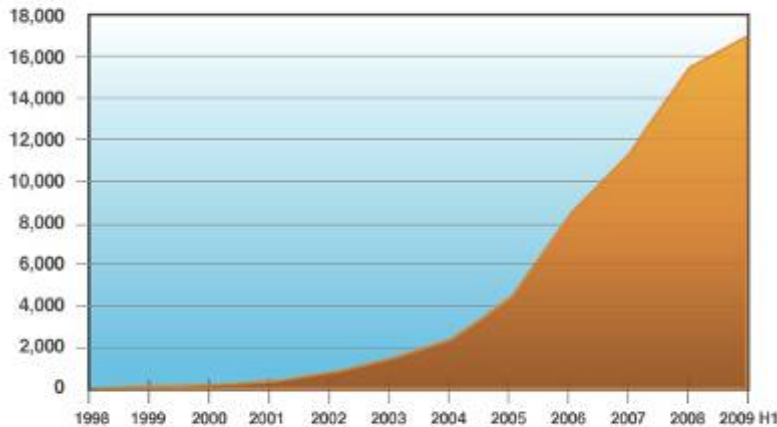
Web App Vulnerabilities Continue to Dominate

- ▶ **49%** of all vulnerabilities are Web application vulnerabilities
- ▶ SQL injection and Cross-Site Scripting are neck-and-neck in a race for the top spot
- ▶ **90%** of injection attacks are attributed to SQL-related attacks
- ▶ Automated toolkits continue to flourish in 2009
- ▶ SQL injection attacks continue to grow up **50%** in Q1 2009 vs. Q4 2008 and nearly doubling in Q2 vs. Q1

Web Application Attacks by Category

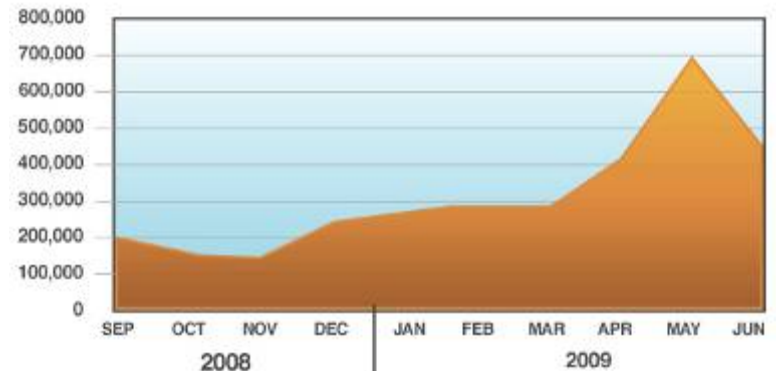


Vulnerability Disclosures Affecting Web Applications
Cumulative, year over year



source: IBM X-Force®

SQL Injection Attacks
Average Daily Attacks by Month



source: IBM X-Force®

source: IBM X-Force®

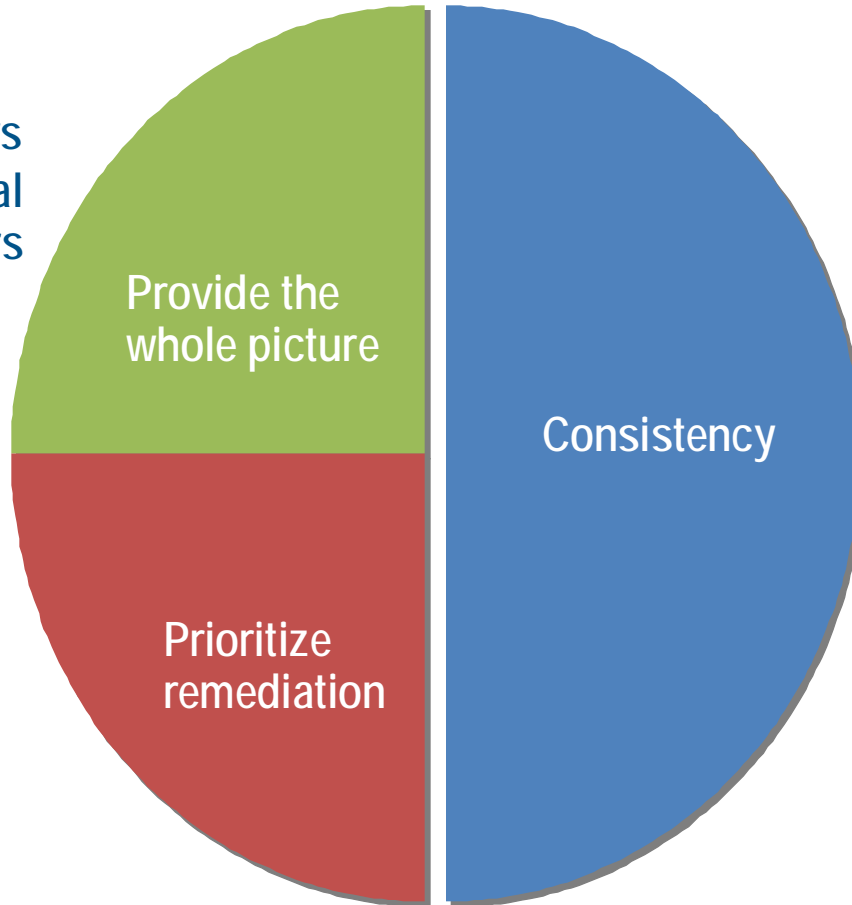


Is this really necessary?

- What's being done isn't working
- Security isn't always included
 - ▶ Software Engineer vs. Software Security Engineer
 - ▶ Coding guidelines vs. *secure* coding guidelines
- “Build Security In” – sounds deceptively simple
 - ▶ Starts with training. This class is a good place to start.
 - ▶ Requires a commitment to change. If we agree that what we are doing isn't working than it should be obvious that we need to change what we are doing.
- Policy is not a four letter word
 - ▶ Requirements, Requirements, Requirements
 - ▶ Developers need security requirements, if not given don't assume, demand.
- Have a plan before you need one

Follow the path to secure coding

Large-scale design flaws typically trump individual coding errors



Developers must identify all vulnerabilities in the code, then remediate the greatest risks first

Create consistent processes, policies, and a culture of improved security

Sometimes the answers can **only** be found in the source code

- Does the application enforce or even use appropriate access controls?
- In what ways and in what places does the application attempt to connect to the network?
- Is there malicious code or back doors in your applications?
- Can user inputs or outputs can corrupt your system ?
- Is customer credit card information encrypted?
- Is sensitive data being stored outside of your database?

```

/**
 * Gets the instructions attribute of the sqlInjection object
 * @return The instructions value
 */
protected String getInstructions(WebSession s)
{
    String instructions = "Enter your account number to reviv
    return ( instructions );
}

/**
 * Gets the menuItem attribute of the DatabaseFieldScreen objec
 * @return The menuItem value
 */
protected Element getMenuItemC
{
    return makeMenuItemC
    EEN,
}

/**
 * Gets the ranking att
 * @return The rankin
 */
protected Integer getRank
{
    return new Integ
}

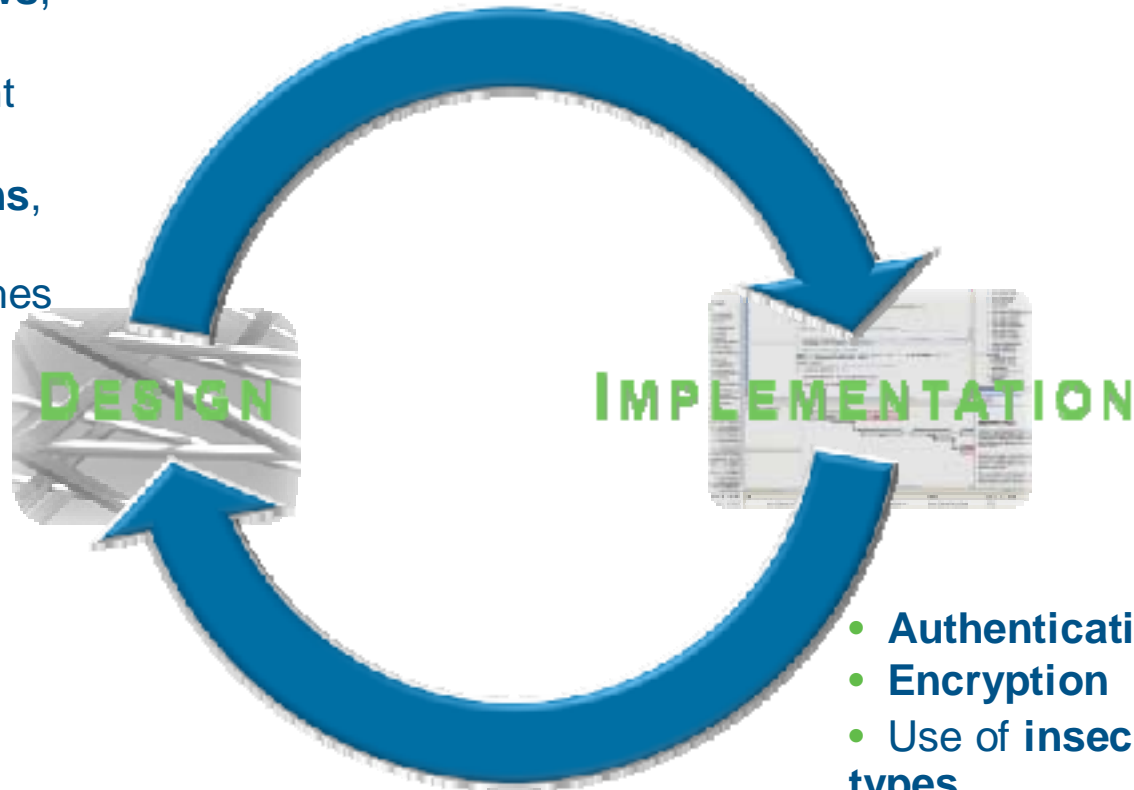
/**
 * Gets the title attribute
 * @return The title value
 */
public String getTitle()
{
    return ( "How to Perform SQL Injection" );
}

/**
 * Constructor for the DatabaseFieldScreen object
 * @param s Description of the Parameter
 */
public void start( WebSession s )
{
    try
    {
        setup( s );
        if ( connection == null )
    }
}
    
```



Where to look for vulnerabilities

- **Buffer overflows**, result from mismanagement of memory
- **Race conditions**, result from call timing mismatches



- **Authentication**
- **Encryption**
- Use of **insecure external code types**
- Validation of data input and application output

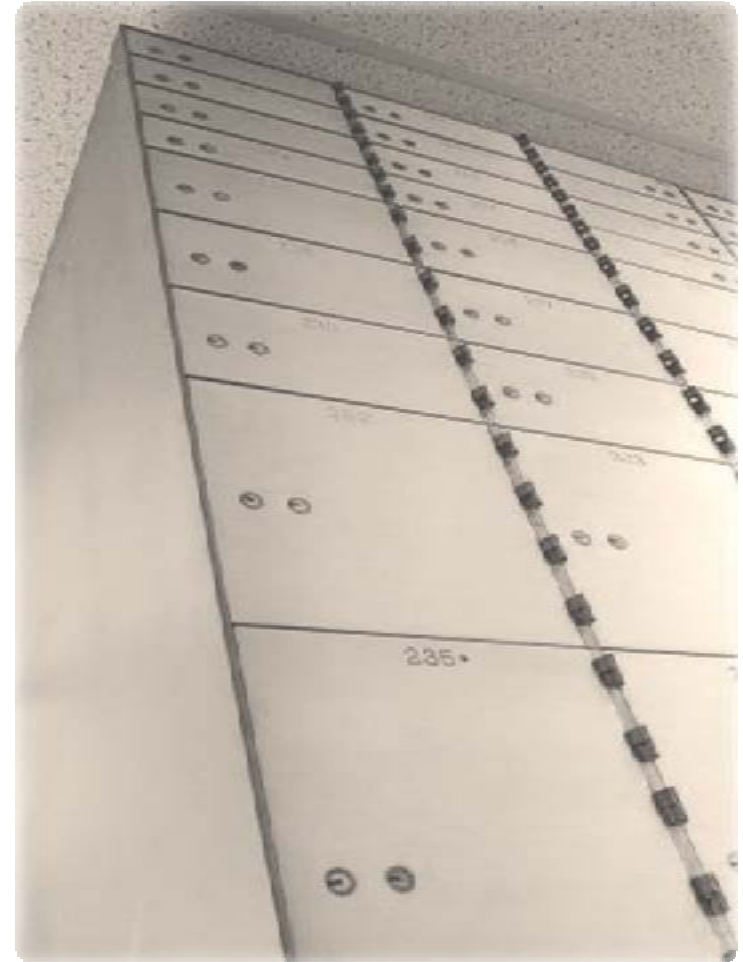
How to look for vulnerabilities

- Manual Code Review
 - ▶ Time-consuming, expensive, error prone

- Penetration Testing
 - ▶ Useful but can only discern a small sub-set possible errors

- Automated Testing Tools

“The most effective approach is to integrate source code vulnerability scanners into the application development, integration and test process.” (Gartner)



An Interesting Aside

From Microsoft's SDL <http://msdn2.microsoft.com/en-us/library/ms995349.asp>

- **“However, one finding will come as no surprise to long-time security researchers: penetration testing is not the way to achieve security. Penetration testing is an element of the Final Security Review (FSR) for a major software release, but product team activities throughout the entire lifecycle focus on threat modeling, code reviews, the use of automated tools, and fuzz testing rather than penetration testing. The latter measures are much more thorough in preventing or removing security bugs than the classic ad hoc penetration testing.”**



That wasn't really helpful

- It is much more effective to look at the places in the SDLC that you can reduce risk.
 - ▶ Requirements
 - ▶ Design
 - ▶ Implementation
 - ▶ Test
 - ▶ Deployment



Requirements

- Identifying security requirements are an integral part of the software design process, and the most neglected
 - Just as good project requirements requires use cases, good security requirements require abuse cases
 - Must be able to identify all potential assets at risk and outline the required and acceptable mitigation requirements.
- ❌ Example of a bad requirement:
- All sensitive data needs to be encrypted
- ✅ Example of a better requirement:
- All sensitive data needs to be encrypted both in transit and at rest using no less than 256 bit AES encryption, see addendum A for the list of items that are considered sensitive for this application.



Design: Policy Definition

- It is important that organizations begin to formalize secure coding guidelines.
- Avoid the temptation to “grade” an organization, development manager, or individual contributor’s, ability to deliver secure code without letting them know what is on the test.
- Policy, in the case of security requirements, is to remove ambiguity as much as possible.
- Examples
 - ▶ New development projects using C/C++ must avoid the use of all following api’s: gets(), strcpy(),unbounded use the printf and sprintf family of calls etc.
 - ▶ All data transferred from web clients that contain customer specific information must be transported using SSL, and if any personal information is stored using cookies the entire application needs to be SSL enabled.



What Details: Threat Modeling

- Threat modeling is an important aspect in developing good security requirements as well as designing good mitigation strategies
- Aspects of threat modeling should occur in several phases of the SDLC
 - ▶ During Requirements
 - Phase 1: Identifying assets at risk and business objectives
 - Phase 2: Generate use and abuse cases
 - ▶ During Design
 - Phase 3: Identify components responsible for controlling access to and from assets identified in Phase 1.
 - Phase 4: Identify the threats posed by Phase 2 against the components outlined in Phase 3.
 - ▶ During Implementation & Test
 - Phase 5: Review application to identify weaknesses against the threats identified in Phase 4 about and review mitigation and remediation efforts.
- Additional resources
 - ▶ <http://msdn2.microsoft.com/en-us/security/aa570411.aspx>
 - ▶ <http://www.projects.ncassr.org/threatmodeling/>



Design: Security Design Review

- The security design review is a critical step in the SDLC. The primary goal of this step is to verify that the policies identified in the requirements and phases 1-4 of the threat modeling exercise have the appropriate mitigation strategies identified in the application architecture.
- Identify any gaps, this may include identifying new threats.
- This should be done as early in the process as possible, for an agile development process every feature iteration that impacts security as identified by the requirements needs to perform this step.



Application Vulnerability Assessment

- Think of this as the verification step. This is to verify that all policy requirements and threats have the appropriate mitigation in the final product.
- This also enables unintended or new threats to have another chance of being found prior to deployment.
- Leverage tools as much as possible to reduce costs.



What To Look For: The Checklist

- ✔ Security-related functions
- ✔ Input/Output validation and encoding errors
- ✔ Error handling and logging vulnerabilities
- ✔ Insecure Components
- ✔ Coding errors

“Detecting and correcting security vulnerabilities early in the application development life cycle, prior to deployment and operations, results in significant risk and cost reduction.”

Gartner

```

/**** Gets the instructions attribute of the DatabaseFieldScreen object
/**** @return The instructions value
/**** @param s Description of the Parameter
/**** @return The instructions value
protected String getInstructions()
{
    return C.instructions;
}

/**** Gets the menuitem attribute of the DatabaseFieldScreen object
/**** @return The menuitem value
/**** @param s Description of the Parameter
/**** @return The menuitem value
protected Element getMenuitem()
{
    return makeMenuItem("SQL Injection Scanner",
        new ImageIcon("resources/scan.png"),
        new ImageIcon("resources/scan.png"));
}

/**** Gets the ranking attribute of the DatabaseFieldScreen object
/**** @return The ranking value
/**** @param s Description of the Parameter
/**** @return The ranking value
protected Integer getRanking()
{
    return new Integer(10);
}

/**** Gets the title attribute of the DatabaseFieldScreen object
/**** @return The title value
/**** @param s Description of the Parameter
/**** @return The title value
public String getTitle()
{
    return C.title;
}

/**** Constructor for the DatabaseFieldScreen object
/**** @param s Description of the Parameter
/**** @return The DatabaseFieldScreen object
public void start( WebSession s )
{
    try
    {
        setup( s );
        if ( C.connection == null )
    }
}
    
```



Security-related Functions



- Weak or Nonstandard Cryptography
- Non-Secure Network Communications
- Access Control Vulnerabilities

- ✓ **MD5 is no longer considered secure for highly sensitive and business critical applications, SHA1 is also considered broken though no practical attacks have been identified.**

“Microsoft is banning certain cryptographic functions from new computer code, citing increasingly sophisticated attack”, <http://www.eweek.com/article2/0,1759,1859751,00.asp>

- ✓ **The following example was from a content management systems password reset function.**

```
/**
 * Generates a random 10 characters password.
 *
 * @return the generated password.
 */
public static synchronized String generate()
{
    return Long.toString(Math.abs(random.nextLong()) % MAX_RANDOM_LENGTH, Character.MAX_RADIX);
}
```

The biggest failure in encryption is not often the algorithm used but more often than not it is the failure to properly identify what data needs to be encrypted and making sure that the appropriate encryption is always utilized.



Input/Output Validation and Encoding Errors



- SQL Injection Vulnerabilities
- Cross-Site Scripting Vulnerabilities
- OS Injection Vulnerabilities
- Custom Cookie/Hidden Field Manipulation

Have we not learned to **NEVER** trust the user, all input needs to be validated?

What is the problem with the code below?

```
public void doGet ( HttpServletRequest req, HttpServletResponse res )
    throws IOException, ServletException
{
    String pageName = getParameter("pageName") != null ? "" :
getParameter("pageName");
    log.info("Request for page: "+pageName);
    String forward = "/" + pageName + "?" + req.getQueryString();
    RequestDispatcher disp = req.getRequestDispatcher(forward);
    disp.forward( req, res );
}
```

It is not all about SQL Injection and XSS (though those are still a huge problem).



Error Handling & Logging Vulnerabilities



Insecure Error Handling
Insecure or Inadequate Logging

Consider the following code example:

```
public void doPost( HttpServletRequest req, HttpServletResponse res )
    throws IOException, ServletException
{
    RequestDispatcher disp = null;
    String user = getParameter("user") != null ? "" : getParameter("user");
    String pwd = getParameter("pwd") != null ? "" : getParameter("pwd");
    if(!validUser(user,pwd)) {
        log.warn("Invalid login received from: " + user + " password:" + pwd);
        disp = req.getRequestDispatcher("/jsp/invalidLogin.jsp");
    } else {
        log.info("Successful login attempt from: " + user);
        disp = req.getRequestDispatcher("/jsp/loginSuccess.jsp");
    }
    disp.forward( req, res );
}
```

There really are two major issues with logging:

1. **Lacking a consistent logging framework.**
 2. **Logging the wrong data or breaking company policy and regulations (think: PCI)**
-



Insecure Components



Unsafe Native Methods

Unsupported API

Improper Use of 3rd Party Application Frameworks

Developers need to understand where the utilities provided by the framework begin and end when related to security. Consider the following code from a .NET web application.

```
<head>
  <title>Registration Form Please Sign-In</title>
</head>
<% String loader = Request.Params["loader"]; %>
<body onload = "<%=loader%>" >
  ...
</body>
```

Even if you have Microsoft's page validation enabled (the default) you are still vulnerable.


As we focus our efforts to fix the low hanging fruit, the attacks are moving to the application layer.

There are many undocumented APIs that exist as public interfaces in the JDK or the .NET framework

Many of these interfaces may bypass internal member data validation that if used directly could crash the JVM (or lead to more serious vulnerabilities <http://www.blackhat.com/presentations/win-usa-03/bh-win-03-schoenfeld.pdf>)



Coding Errors



- Buffer Overflow Vulnerabilities
- Format String Vulnerabilities
- Denial of Service Errors
- Race Conditions

Use of native libraries (System.loadLibrary, [DllImport]) can also expose your web application to this more traditional style of attack.

What's wrong with this code?


```
protected void doGet (HttpServletRequest request, HttpServletResponse response) {
    InputStreamReader inStr = new InputStreamReader(request.getInputStream());
    BufferedReader in = new BufferedReader(inStr);
    while(in.readLine() != null) {
        //process the request
        ...
    }
}
```


Most modern day web applications are immune to the more traditional “overflow” style of attacks, but anytime the user is able to control data that reaches an internal system the possibility exists.


http://documents.iss.net/whitepapers/IBM_X-Force_WP_final.pdf





Follow The Path: The Checklist

- 
 - **Security-related functions**
 - Weak or Nonstandard Cryptography
 - Non-Secure Network Communications
 - Application Configuration Vulnerabilities
 - Access Control Vulnerabilities

- 
 - **Input/Output validation and encoding errors**
 - SQL Injection Vulnerabilities
 - Cross-Site Scripting Vulnerabilities
 - OS Injection Vulnerabilities
 - Custom Cookie/Hidden Field Manipulation

- 
 - **Error handling and logging vulnerabilities**
 - Insecure Error Handling
 - Insecure or Inadequate Logging

- 
 - **Insecure Components**
 - Unsafe Native Methods
 - Unsupported Methods
 - Improper use of 3rd Party Application Frameworks

- 
 - **Coding errors**
 - Buffer Overflow Vulnerabilities
 - Format String Vulnerabilities
 - Denial of Service Errors
 - Race Conditions



Where?

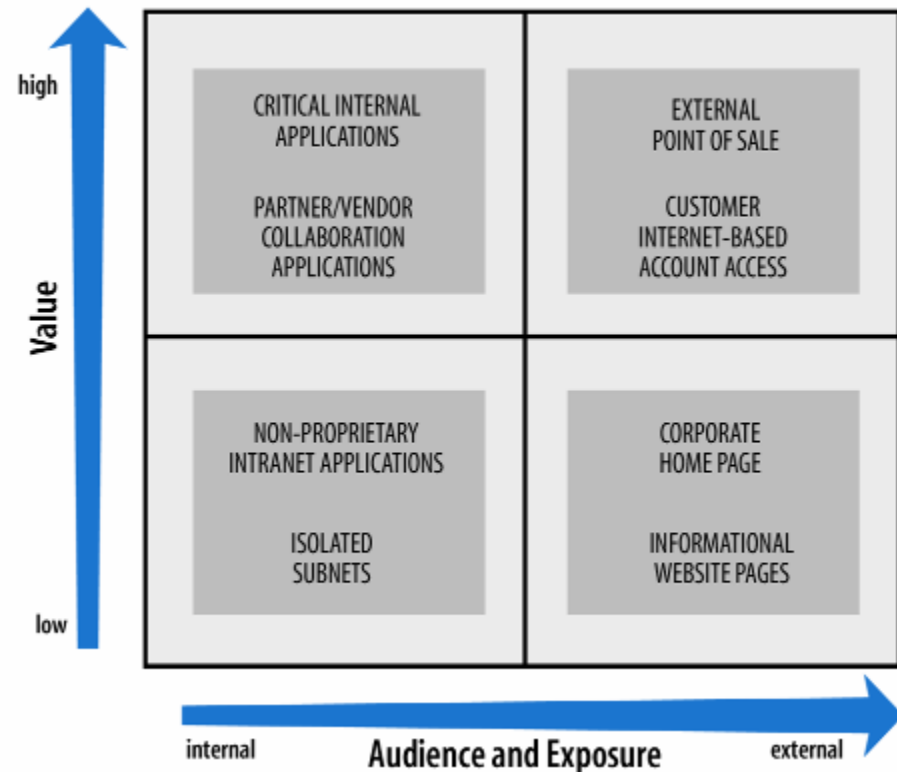
- **Baking security into requirements:** gathering security requirements/needs, abuse cases, and threat modeling
- **Baking security into design:** security design patterns, security reviews and threat modeling
- **Baking security into development:** secure coding guidelines, tools, and audit
- **Baking security into testing:** negative testing, thinking like the bad guy and “red teams”
- **Baking security into deployment:** secure deployment guidelines, secure update mechanisms (patching) and much, much more!



When?

- As often as is practical
 - Prioritize the most critical applications
 - Separate legacy from new development
 - Customer facing vs. internal

FIGURE 1: VALUE AND EXPOSURE

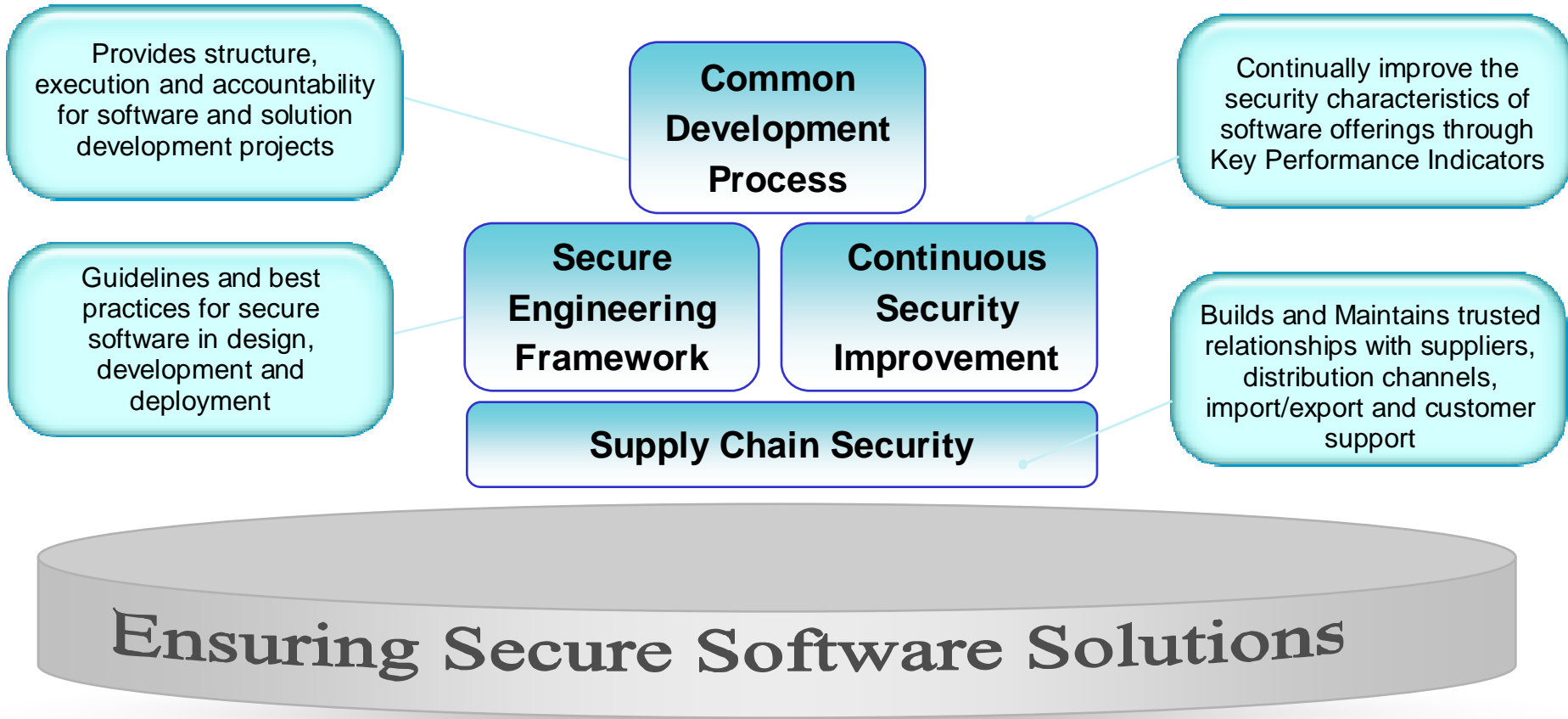


How: Objectives for Practical Security

- Improve *existing* development process, not create new one
- Maximize security impact of personnel and technologies
- Use models as initial framework and tailor to individual organization
- Select model with consideration for future requirements



IBM Secure Engineering Initiative



Link to Security Engineering Framework:
<http://www.redbooks.ibm.com/redpieces/abstracts/redp4641.html?Open>

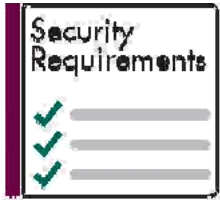


SecureDevelopment Framework

	Inception	Elaboration	Construction	Transition
Secure Development Framework Goal				
Activities	<ol style="list-style-type: none"> 1.1. Engage Security Expert 1.2. Determine Predictive Threat Index <ul style="list-style-type: none"> 1.2.1. High¹ 1.2.2. Medium² 1.2.3. Low 1.3. Map user requirements to security requirements 1.4. Determine authorization requirements 1.5. Identify key internal/external compliance objectives 1.6. Define application security test process & deliverables 1.7. Adjust project plan to include security resources 1.8. Contract secure code review (5.1) 1.9. Contract manual pen test (5.2) 	<ol style="list-style-type: none"> 2.1. Architecture Review 2.2. Identify secure design techniques 2.3. Identify certified components responsible for security functions 2.4. Document attack surface 2.5. Create threat modeling document 2.6. Review/modify security requirements 2.7. Identify components for Secure Code Review 2.8. Define secure integration with external systems 2.9. Define security test requirements 2.10. Determine authorization requirements model 2.11. Update Security Master Test Plan 2.12. Update test schedule and budget 	<ol style="list-style-type: none"> 3. Code / Build <ul style="list-style-type: none"> 3.1. Certified components 3.2. Static Code Assessment 3.3. Dynamic Application Assessment 4. Test / Verify <ul style="list-style-type: none"> 4.1. Peer Code Review 4.2. Static Code Assessment 4.3. Dynamic Application Assessment 	<ol style="list-style-type: none"> 5. Final Security Review <ul style="list-style-type: none"> 5.1. Secure Code Review 5.2. Manual penetration testing 5.3. Static Code Analysis 5.4. Dynamic Application Assessment 5.5. Review of all bugs for possible security vulnerabilities 5.6. Review threat model for possible late developing threats
Deliverables	<ul style="list-style-type: none"> - Security Expert assigned - Predictive Threat Index - Preliminary security requirements defined - Security test strategy 	<ul style="list-style-type: none"> - Architecture Review - Minimized application attack surface - Application security test roles - Threat Model - Security requirements in well defined components - Application security test plans - Certified components identified 	<ul style="list-style-type: none"> - Working application 	<ul style="list-style-type: none"> - Problems, defects, enhancements logged - Detailed test results - Validated requirements - Updated test results in centralized location - Certification
Tools	<ul style="list-style-type: none"> - Security Knowledge Portal - Security Consultant - Design Review Checklist - Requirements Software - Predictive Threat Index calculator 	<ul style="list-style-type: none"> - Security Knowledge Portal - Architectural Review Checklist - Threat Modeling Software - Platform dependent coding checklist - Certified Components 	<ul style="list-style-type: none"> - Security Knowledge Portal - Static Code Analyzer - Dynamic Application Analysis Tool - Certified Components - Security Development Guidelines 	<ul style="list-style-type: none"> - Security Knowledge Portal - Security Auditor - Static Code Analyzer - Dynamic Application Analysis Tool - Final Review Checklist



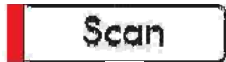
Core Responsibilities



Set security requirements: A manager or security expert defines vulnerabilities and how to judge criticality



Configure analysis: The source code analysis tool is customized to address internal policies.



Scan source code: The analysis tool is run against the target application to pinpoint vulnerabilities.



Triage results: Security-minded staff study results to prioritize remediation workflow.



Remediate flaws: Vulnerabilities are eliminated by rewriting code, removing flaws, or adding security functions.



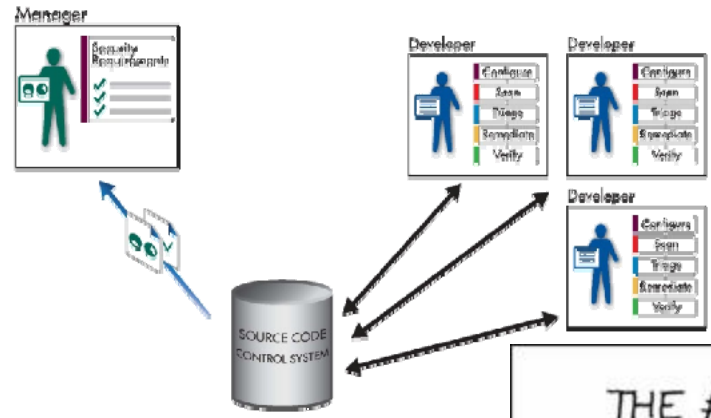
Verify fixes: The code is rescanned to assure vulnerabilities are eliminated.



Model I: Independent

Benefits

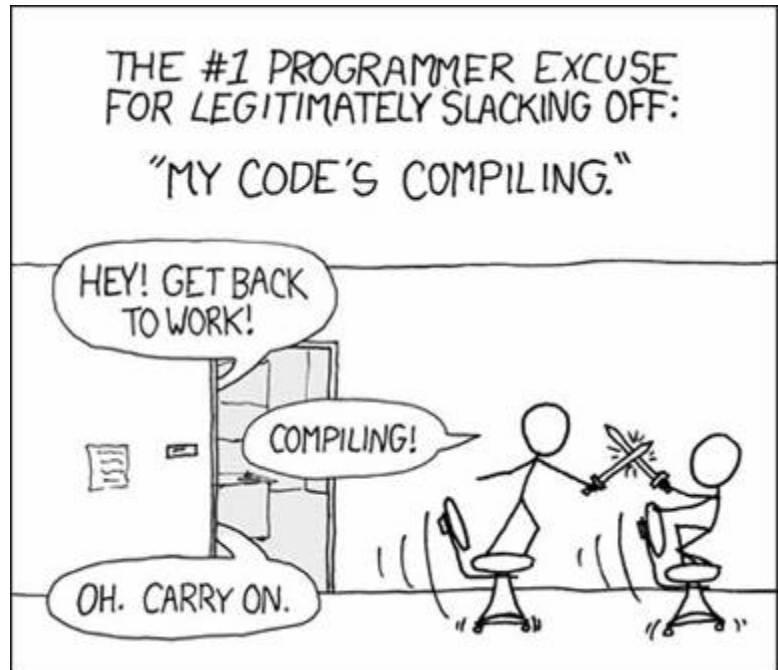
- Developer-only implementation
- Low initial investment



<http://xkcd.com/303/>

Challenges

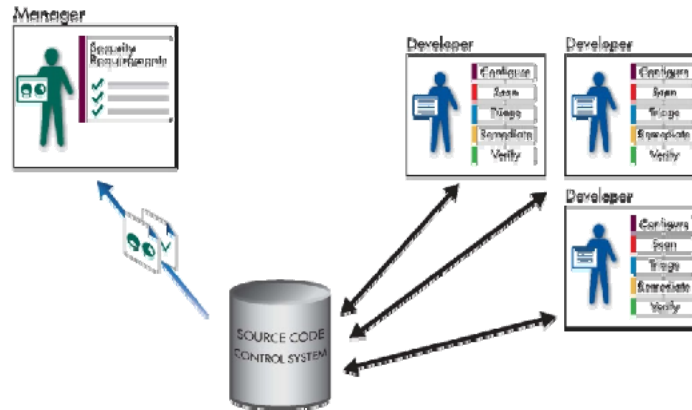
- Scalability
- Redundancy of work
- Reporting/tracking progress
- Enforcing policy
- Expertise requirements



Model I: Independent

Benefits

- Developer-only implementation
- Low initial investment



Challenges

- Scalability
- Redundancy of work
- Reporting/tracking progress
- Enforcing policy
- Expertise requirements

Best Practices

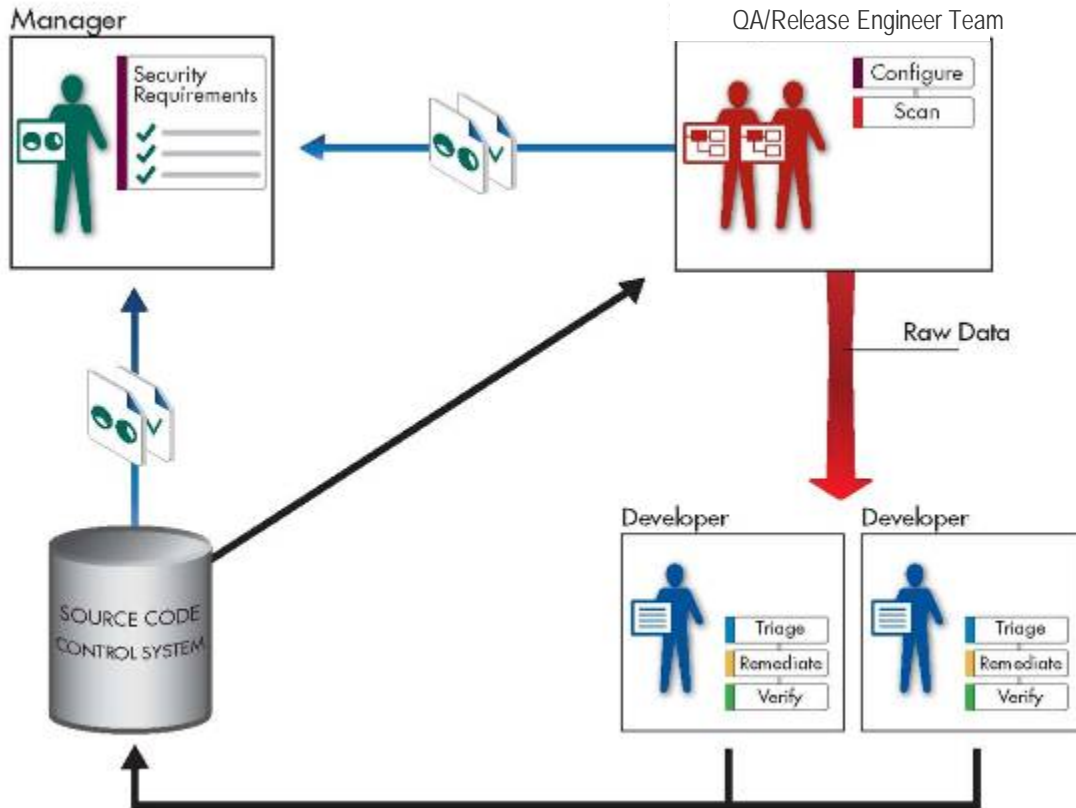
- Establish centralized security requirements
- Conduct security reviews among developers
- Establish a security-capable developer as a mentor



Model II: Distributed

Managers

- Refine security requirements
- Track development progress and review assessment data



QA/Release Engineer Team

- Configure scanner for build integration
- Sync code at each milestone
- Scan entire app with new milestone components
- Provide raw data to development

Developers

- Triage analysis results
- Perform necessary remediation
- Verify fixes before checking code back in



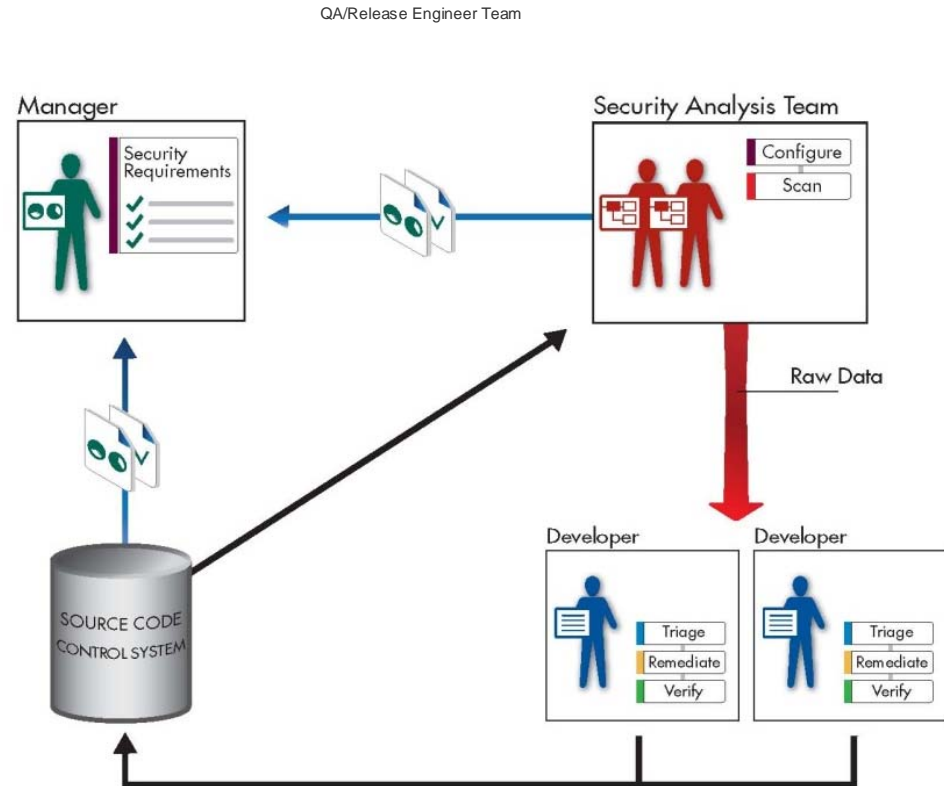
Model II: Distributed

Benefits

- Deployment flexibility
- Integration with build environment
- Policy enforcement
- Central reporting

Challenges

- Scalability
- Redundancy of work
- Expertise requirements



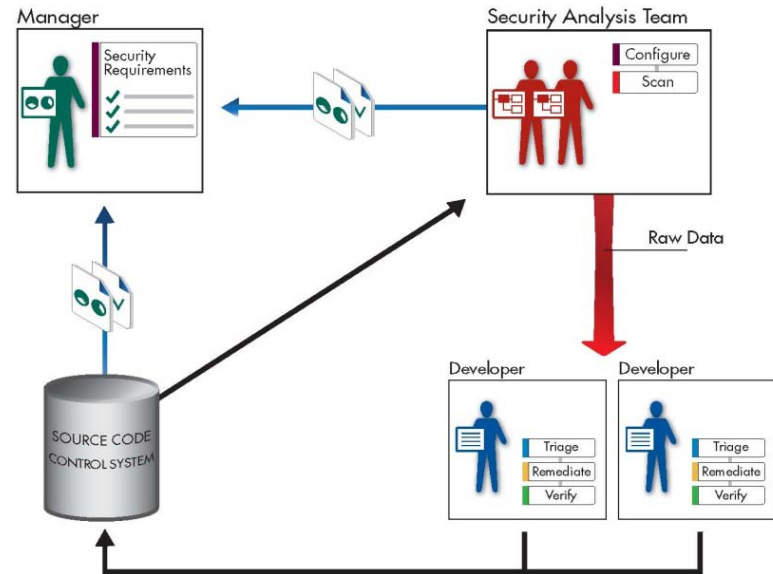
Model II: Distributed

Benefits

- Deployment flexibility
- Integration with build environment
- Policy enforcement
- Central reporting

Challenges

- Scalability
- Redundancy of work
- Expertise requirements



Best Practices

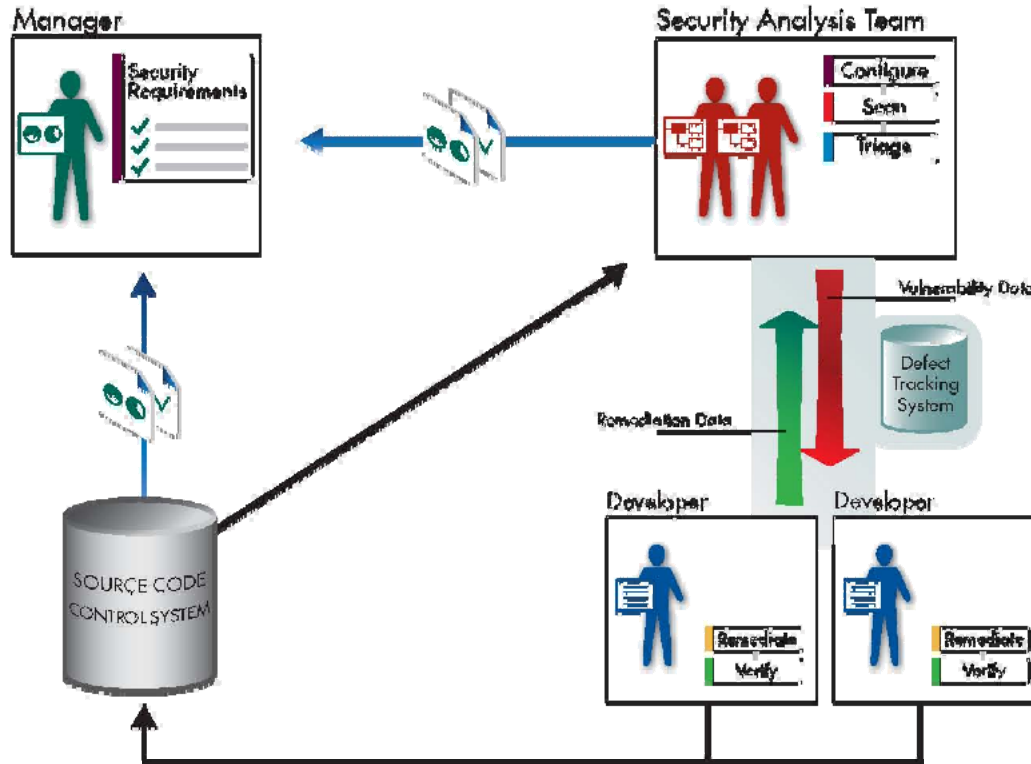
- Scan early in life cycle
- Assign developers specific components to review
- Establish a security-capable developer as a mentor



Model III: Centralized

Security Analysis Team

- Configure scanner for build integration
- Retrieve code for analysis
- Scan entire application
- Triage results
- Assign vulnerabilities to developers



Managers

- Refine security requirements
- Track development progress, review vulnerability data, and monitor remediation results

Developers

- Perform necessary remediation
- Check in code



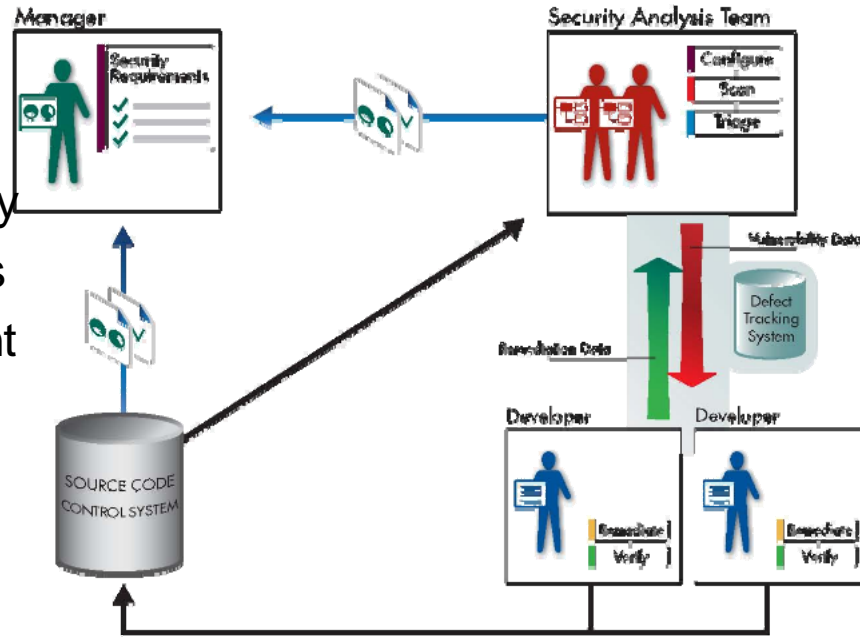
Model III: Centralized

Benefits

- Deployment flexibility
- Separation of duties
- Central management
 - Long-term value

Challenges

- Resource requirements
- High-level commitment



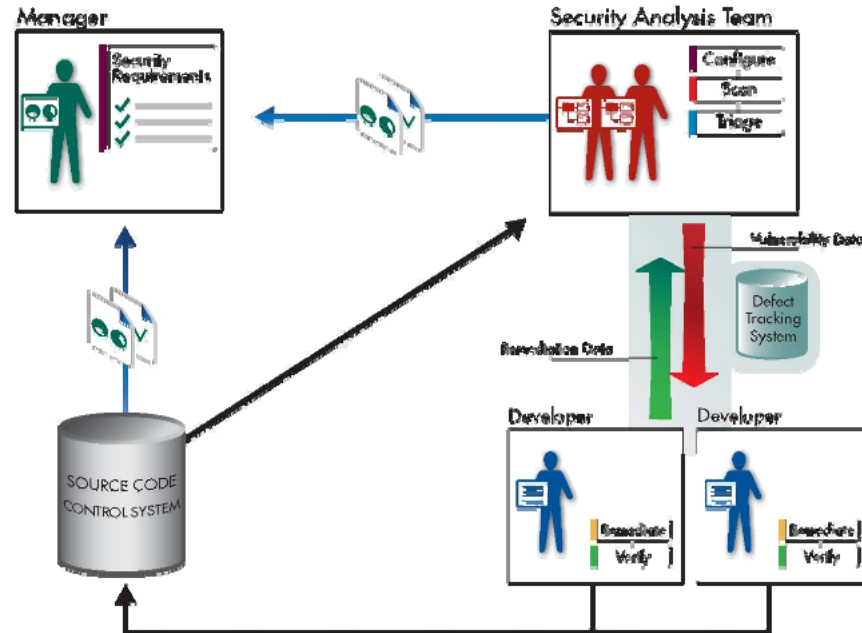
Model III: Centralized

Benefits

- Deployment flexibility
- Separation of duties
- Central management
- Long-term value

Challenges

- Resource requirements
- High-level commitment



Best Practices

- Plan process with development and security
- Focus on training and guidance for security team
- Integrate with existing technologies



Benefits	Independent Model	Distributed Model	Centralized Model
Effectively reduces security vulnerabilities	✓-	✓	✓+
Centralized management and remediation strategy		✓	✓+
Centralized analysis configuration		✓	✓
Cross-enterprise reporting of results and progress		✓	✓+
Supports distributed development teams	✓-	✓	✓
Supports small development and audit projects	✓+		
Prioritized remediation assigned to developers			✓
Low level of management commitment	✓		
Scales to large applications and development teams		✓	✓+



For More Information...

To learn more about using IBM's secure engineering framework

<http://www.redbooks.ibm.com/redpieces/abstracts/redp4641.html?Open>



Questions



Daily iPod touch giveaway

- Complete your session surveys online each day at a conference kiosk or on your Innovate 2010 Portal!
- Each day that you complete all of that day's session surveys, your name will be entered to win the daily IPOD touch!
- On Wednesday be sure to complete your full conference evaluation to receive your free conference t-shirt!

SPONSORED BY

AllianceTech
Intelligent EVENTS





www.ibm.com/software/rational

© Copyright IBM Corporation 2010. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

