**Companies that reduce vulnerabilities in applications before they are shipped or deployed are recognizing tremendous cost savings internally and for their customers, partners, and other stakeholders that rely on their software.**
December 2009

Rational® software

# Secure at the Source

***Implementing source code vulnerability testing in the software development life cycle***

*Ryan Berg*
*IBM Senior Security Architect*
*IBM Software Group*

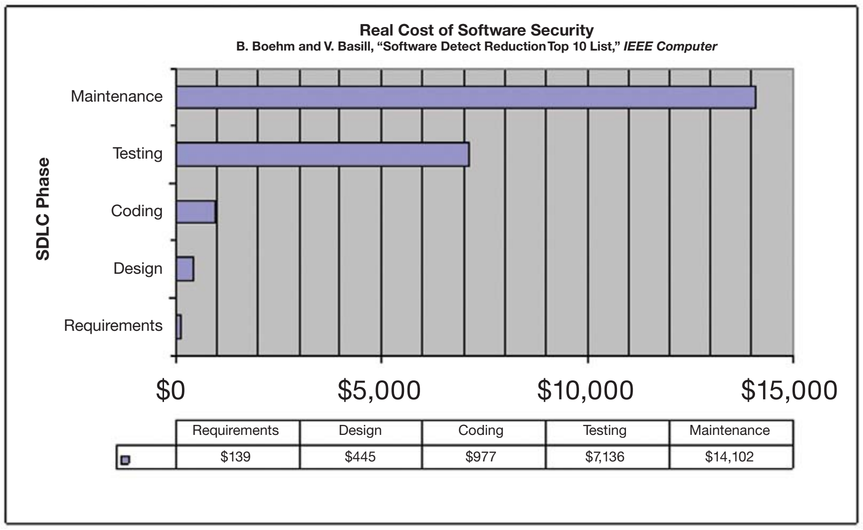
Countless studies and analyst recommendations suggest the value of improving security during the software development life cycle rather than trying to address vulnerabilities in software discovered after widespread adoption and deployment. The justification is clear.

For software vendors, costs are incurred both directly and indirectly from security flaws found in their products. Reassigning development resources to create and distribute patches can often cost software vendors millions of dollars, while successful exploits of a single vulnerability have in some cases caused billions of dollars in losses to businesses worldwide. Vendors blamed for vulnerabilities in their product's source code face losses in credibility, brand image, and competitive advantage. A study in 2005 by Carnegie Mellon found that the stock price of vendors declined an average of .63 percent compared to the NASDAQ® after a vulnerability is discovered in their software.[1]

Studies with this level of detail are not available for flaws found in custom enterprise software developed in-house or outsourced. There is agreement that the earlier in the life-cycle flaws are discovered the cheaper they are to address. As shown in the chart 1 research published by B. Boehm and V. Basali in IEEE Computer© found that fixing a software defect after deployment costs more than 100 times what it would have cost to fix it at the first stages of the development life cycle.[2] For security defects, late-stage costs are often much higher, because in addition to having to remediate the flaws, successful exploits might lead to data theft, sabotage, or other attacks.

*Finding and fixing security issues early is an application project can help reduce development costs while improving software quality. Source code security tools implemented and used across the software development life cycle are known to provide such results.*

**Companies that reduce vulnerabilities in applications before they are shipped or deployed are recognizing tremendous cost savings internally and for their customers, partners, and other stakeholders that rely on their software.**



| | Requirements | Design | Coding | Testing | Maintenance |
|---|---|---|---|---|---|
| ▨ | $139 | $445 | $977 | $7,136 | $14,102 |

Automated source code analysis is widely recognized as the most effective method of security testing early in the life cycle, because it allows assessments of any piece of code without requiring a completed application. The best of these technologies provide the most valuable results by pinpointing the vulnerability at the precise line of code and detailing information about the type of flaw, degree of criticality, and how to fix it. Ethnical hacking is also an important element of software security, but its value comes later in the life cycle, when it can be used on a completed application with a functional interface.

## Roadblocks to building in security

Among the hurdles that might impede security testing in the software development life cycle, the largest is typically the gap between development and security. The skill sets themselves are rarely present in the same individual or even group, and organizationally. There is very little inherent synergy. While development goals focus on product functionality and on-schedule delivery, security analyst are often tasked with eliminating vulnerabilities and implementing security controls only after the applications are completed and deployed. To effectively decrease vulnerabilities created during the development process, cooperation must be achieved between development and security teams. In all cases, higher-level management support for improving security during development is essential.

In addition to organizational impediments, a general hesitancy to change or revise an existing software development life-cycle process might delay implementation of security testing. However, a simple understanding of the business-level benefits to be gained is usually enough incentive to move things forward. Similar to this conceptual roadblock, there are many misconceptions about integrating security analysis during development that must be overcome before an initiative can move forward.

**Fiction:**
The development schedule cannot afford to be stretched any further, not even to address security issues.

**Fact:**
There might be initial lapses in the development cycle, especially as individuals learn the new system. However, this is the most time-efficient method for reducing software risk. The process eventually reduces development time by instilling good secure coding practices among developers. The only faster alternative is to do nothing to improve software security. That is an option that most organizations certainly cannot afford in the long term.

**Fiction:**
We are already doing peer review; therefore, we do not need additional security code reviews.

**Fact:**
A peer review is not a substitute for a security review. Peer reviews are typically used to find functional bugs. Unless reviewers have a deep understanding of application security, many of the more critical security vulnerabilities and design flaws are missed. In many cases, the best-intentioned user requirement implemented without functional error can lead to the greatest security risk.

## Core responsibilities

Many enterprises still find it challenging to identify the most appropriate method and resources to implement source code analysis in their development life cycle. The three models explained in this paper represent common scenarios currently being used to successfully reduce vulnerabilities during development. These models help establish criteria for assessing goals, resources, obstacles, and ultimately, the most favorable approach for individual organizations.

The purpose of this paper is not to describe a new threat modeling process. Rather it is to document a series of workflow models to help guide how automated source code scanning can be implemented into an existing development process.

Although it is clear that development organizations and processes each have their own distinct characteristics, the models outlined address the common elements that must be leveraged to achieve effective security testing. In the descriptions below, the primary functions that must be served by existing staff or experts brought in during implementation are:

**Set security requirements:** A manager or central source of security expertise defines the vulnerabilities and how to judge criticality based on business needs.

**Configure analysis:** Internal definitions are used to customize the source code analysis tool to match policies.

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

**Prioritize results:** Staff members with knowledge of security and of the application study results to prioritize remediation workflow to address high-risk security flaws first.

**Remediate flaws:** Vulnerabilities are eliminated by rewriting code, removing flawed components, or adding security-related functions.

**Verify fixes:** The code is rescanned and studied to assure the code changes have eliminated the vulnerability while maintaining application functionality.

## I: Independent model

This is the model most often considered when the concept of source code analysis is first introduced. In the independent model, each developer is responsible for analyzing code for security vulnerabilities, identifying the most critical, providing necessary fixes, and verifying that the flaws have been eliminated. Depending on the organization and the application requirements, these security efforts are typically practiced at regular intervals during the development cycle. Software engineers scan their own code after making changes or additions and remediate any vulnerability before checking files back into the source code control system. Ideally, individuals scan the entire application to analyze their code in context to track data flow and potential flaws across the complex interactions of various files and functions. Unless the code base is relatively small, this might be extremely time-consuming.

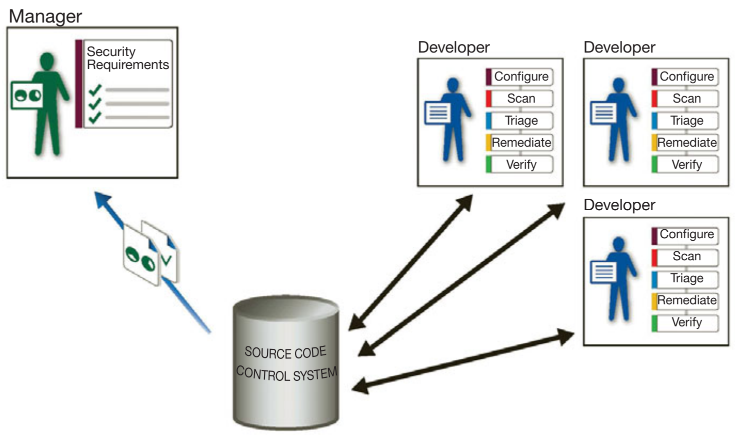
### How does it work?

This model is assumed to be implemented during the development phase as part of an existing software development life cycle. It requires the developers involved to have sufficient security knowledge not only for identifying vulnerabilities and performing appropriate prioritization, but also for understanding how to best fix them. Some management input is usually required to provide basic security requirements as well as guidance for decision making (for example, how to judge criticality for Web interface vulnerabilities as opposed to those found in a database application). When security policies are not defined by a central source, developers are left to make individual decisions about what constitutes a vulnerability or a fix.

Managers are able to track development progress to measure delays caused by these security efforts. Generally, this model lacks higher-level reporting on vulnerabilities found and fixed because theoretically the vulnerabilities identified are eliminated before entering into a centralized reporting structure.

### When does it work?

Organizations with small development teams and small applications get the most value from the independent model. In these cases, it is much easier to provide sufficient training and guidance so the developers' security efforts result in effective remediation. Although tracking improvements in this model is difficult, it is expected that the developers learn from their security efforts and avoid common mistakes as they write future code. The benefits and the relatively little investment required to initiate the independent model proves to be a feasible method for introducing security into small development organizations. It is also an option for simple system integrator projects. As long as the analysis and remediation is performed by a few developers.



## Workflow:

**Managers**

- Define security requirements

**Developers**

- Checkout code
- Add or modify functionality
- Configure scanner
- Scan application
- Prioritize results
- Perform necessary remediation
- Run scan to verify fix
- Check-in latest code changes

**Managers**

- Review development reports

For development organizations without broad security expertise, a variation of this model is created by singling out an individual or small number of engineers with security backgrounds or access to training. In this approach, the security developers take a mentorship role in identifying and remediating flaws to help their colleagues better understand the basics of secure development.

### When it does not work?

The independent model is not scalable to small applications or small teams that are specialized in source code review. Most organizations cannot rely on developers to have sufficient expertise to make important security decisions and training to achieve that level of understanding often requires an impractical commitment of resources. Even if developers reach this point, the lack of centralized controls and processes often make this an inefficient approach, leading to redundant work among developers who scan the same code bases simultaneously and might potentially work on fixing the same vulnerabilities. This model also fails because of the difficulty in creating and enforcing secure coding policies across more than a few developers. Without quantifiable standards defined and enforced throughout the company (for example, what level of encryption to use or how to validate input), the standard practices become whatever methods are favored by individual reviewers, which leads to inconsistency and in many cases, poor security.

Another shortcoming is the inability to track valuable project data from a business perspective, such as number and type of vulnerabilities discovered, improvement in application security over time, and return on investment for the security resources used. If developers possess the right knowledge and tools, it is very likely that vulnerabilities are eliminated during development. However, without reports and artifacts demonstrating this improvement, there is no viable way to communicate the value to executives, auditors, customers, partners, and other stakeholders.

## Best practices

For best results in the independent model:

Establish a set of quantifiable, enforceable security requirements to guide remediation efforts. Conduct security reviews among developers to verify that all security fixes effectively eliminate vulnerabilities without negatively affecting functionality.

Identify and train a security-capable member of the development team to act as a mentor. This person guides other developers on analysis, prioritization, and security review to verify fixes.

## II: Distributed model

Similar to the independent model, the distributed model relies on the developers to perform the majority of the security functions. The vulnerability scanning is conducted centrally on the complete application. Typically, the quality assurance (QA) or release engineering team assumes this responsibility, which enables this model to integrate easily with existing development structures. The timing and frequency of assessments can be controlled to meet a flexible range of testing requirements, from an Agile development process to a more structured process such as waterfall development.

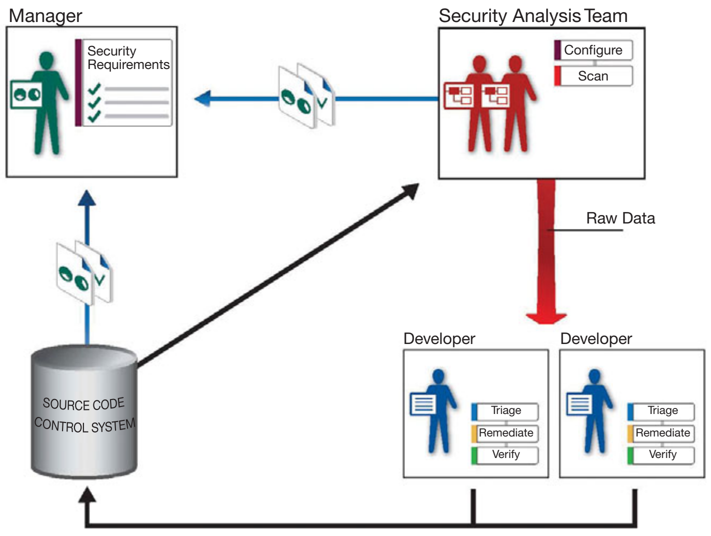
### How does it work?

The security analysis can take place either as a requirement before entering the QA phase or as part of an acceptance test after QA has begun. The QA or release engineering team configures the analysis according to centralized security requirements, scans the entire application, and distributes the results as raw data to the development team. Individual developers are then responsible for analyzing the data to identify the most critical vulnerabilities and performing the appropriate remediation. With this distribution of responsibilities, the development team requires substantial security knowledge and skill. The QA or release engineering team carries out simple configuration and scheduling tasks.

Depending on the size of the application and number of vulnerabilities found in the initial scan, several iterations of security testing might be required in the distributed model to verify that the fixes are effective while maintaining application functionality. Repetitive testing is also critical as the application is being developed. New dependencies and interactions in the code might expose vulnerabilities that had previously eluded discovery. Increasing the frequency of assessments in this model significantly improves the team's ability to discover vulnerabilities early, though a balance must be found in order to adhere to the development schedule. A primary advantage of the distributed model is the ability to find this balance. Adjustments can be made more easily to a centralized scanning process than to individual scanning as in the independent model. This model also eliminates the workflow redundancies that occur when developers are individually responsible for configuring and running their own assessments.

### When does it work?

Medium-sized development teams using a formal software development process are best suited for the distributed model. It functions effectively within an Agile development process because the frequency of testing increases the chances of finding and correcting vulnerabilities early in the life cycle. A waterfall development process offers fewer opportunities, but also benefits from the distributed model because the larger scope of product milestones maximizes the value of each assessment.



### Workflow:

**Managers**

Define security requirements

**QA or Release Engineer team**

Configure scanner for build integration

Sync code at each milestone

Scan milestone components

Provide raw data to development

For cases in which developers do not have sufficient security experience, specialists might be brought into this model to receive results of the analysis and perform the necessary prioritization. They might be at a small disadvantage being unfamiliar with the application architecture, but this approach offers significant freedom for the developers to concentrate on their coding responsibilities. This approach only works if the security audit team is integrated as part of the software engineering team. This model is assumed to be carried out inline with the development life cycle and not as part of a parallel or out-of-band process.

### When does it not work?

The distributed model does not scale well for complex applications, large development teams, or development life cycles that are not driven by functional or component-based milestones. It begins to break down when developers have to prioritize their responsibilities. Developers have the same problems as their counterparts in the independent model, including duplication of work and lack of detailed knowledge of security and the overall application architecture. If the prioritization process does not accurately identify the most critical vulnerabilities or individual developers are working with overlapping components of the code, the value of remediation efforts cannot keep up with the losses in productivity.

### Best practices

For best results in the distributed model:

- Begin scanning early in the life cycle. Maintain a regular and frequent schedule of assessments. Assign developers responsibility for securing distinct components of the application to reduce redundant work as much as possible.
- Identify a security-minded member of the team to act as a mentor, guiding other developers on analysis, prioritization, and peer review to verify fixes.

## III: Centralized model

The centralized model is the most flexible approach. This model can be adapted to any size team, independent of the software development process and application complexity. It is generally not recommended for smaller teams to begin with however, the initial investment in resources required. Source code analysis programs that start with one of the two other models eventually evolve into the centralized model because of the gains in efficiency and measurability of results. This is especially true as application development requirements become more complex and the size of the team increases.

### How does it work?

Unlike the other two models, which require developers to become security experts, the centralized model allows the security functions to be carried out by the group with the greatest experience and knowledge of software vulnerabilities. The security analysis team scans the entire application, leveraging a centralized source of expertise and technology, regardless of whether they exist internally or externally of the development life cycle. Raw results are prioritized by this security team. The information they generate provides developers with a prioritized remediation workflow based on the criticality of vulnerabilities. The team is more likely to properly interpret security requirements when they configure the analysis because they understand the business-level issues of risk management.

Vulnerability remediation assignments are categorized and sent to individual developers through a defect tracking system. This allows the entire team to monitor progress of flaws being fixed.

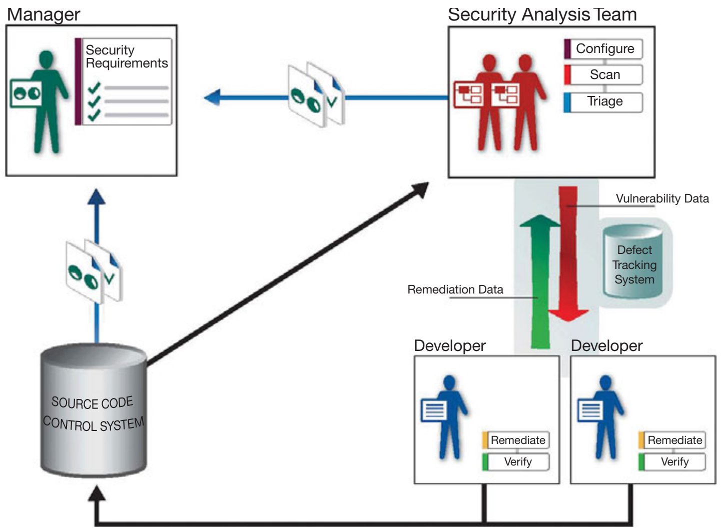
Developers are allowed to focus more of their time on advancements and improvements to the source code, either adding functionality to the software or recoding elements that were considered vulnerable.

The security audit team in this model provides developers with remediation advice that is context specific, but also incorporates remediation guidelines according to corporate policy. For example, the audit team might find an SQL injection vulnerability in the credit card number processing of an online billing application. The recommended fix might be to modify the SQL to use a stored procedure or parameterized SQL statement or use the corporate standard credit card validation routine. The central security team can correctly report this vulnerability and assign remediation based on company-specific policies, instead of requiring developers to make individual decisions.

In this model, developers run their own scans to verify that their fixes are effective. The scope of a vulnerability might only manifest itself when the entire application is analyzed. It is better for the security team to own responsibility for verification.

### When does it work?

The centralized model's primary advantage is the flexibility to integrate efficiently either inside the software development life cycle, as a complement for internal software audit teams, or externally, as a tool for security integrators, or code review services. This model can be used independent of a formalized development process, though its effectiveness might be diminished without a defined structure. Centralizing the configuration, analysis, and prioritization makes this the easiest model to manage. This model has the capability to scale quickly as the development team or project scope increases.



## Workflow:

**Managers**

Define security requirements

**Security Analysis Team**

Configure scanner for build integration

Retrieve code for analysis

Scan entire application

Prioritize results

Assign vulnerabilities to developers

**Developers**

Perform necessary remediation

Check-in code

**Managers**

Track development progress, review vulnerability data, and monitor remediation results

Because of the various deployment scenarios possible with the centralized model (development, internal audit, and external code review) it often requires different delivery methods for the source code. Connecting to the source code control system might work well if the analysis team is inline with the software development life cycle. External review teams request remote access or code delivered on portable storage devices, such as a CD or USB flash drive.

### When does it not work?

The centralized model can be applied to all project scenarios, though it might be more complicated than necessary for small teams or applications. The centralized model requires support and commitment from management to assure that sufficient resources are allocated. It is important that there are common goals defined for both the participating development and security organizations. Without this cross-functional cooperation and dedication of sufficient resources, organizations cannot expect a measurable return on investment.

## Best practices

For best results in the centralized model:

- The security and development teams must be included in planning and sharing information prior to implementation to achieve agreement. Each team must be in agreement on the following key elements:
  - — Application design and architecture
  - — Security requirements
  - — Security training
  - — Prioritization of remediation workflow
- Guidance for the analysis team to assure that their results, prioritization, and vulnerability assignments achieve the great impact
- Integration of the process with existing technologies, such as defect tracking systems and integrated development environments, to make the transition as smooth as possible for staff and to maximize long-term efficiency

## Choosing the right model

The three models all represent current deployment scenarios being used successfully in various development organizations. In each approach, there is a degree of flexibility to accommodate specific requirements of existing processes. The models might ultimately take on different shapes. However, the fundamental stages and functions serve as excellent guidance for organizations interested in implementing security testing during development.

When choosing one of the models as a template, it is important to catalogue existing resources (security expertise, technologies, and service partners) as well project objectives (fewer security patches, competitive advantage, and compliance). Appendix I describes a hypothetical decision-making and implementation process based on factors that actual development organization are facing.

With growing awareness of software vulnerabilities as a critical problem in information security, and with the availability of accurate, efficient source code vulnerability analysis technologies, implementing security testing into software development is occurring more often, and with a greater degree of success. Companies that reduce vulnerabilities in applications before they are shipped or deployed are recognizing tremendous cost savings internally and for their customers, partners, and other stakeholders that rely on their software.

| Requirements | Independent model | Distributed model | Centralized model |
|---|---|---|---|
| Effectively reduces security vulnerabilities | X | X | X |
| Centralized vulnerability and progress reporting | | X | X |
| Analysis integrated with automated build system | | X | X |
| Supports distributed development teams | X | X | X |
| Remediation assigned to individual developers | | | X |
| Ability to prioritize workflow by criteria | | | X |
| Scalable for 50 or more developers | | X | X |
| Supports large applications | | | X |

## Appendix I: Case study using the centralized model

Consider Acme Software, a hypothetical provider of a Web-based billing and fulfillment application, built using the J2EE™ framework. Acme has a team of 25 software engineers, including release or quality assurance (QA), located at its company headquarters in Austin, TX, and an additional outsourced development team of 25 engineers located in New Delhi, India. The engineering team follows an Agile software development process with multiple release and testing phases throughout the software development life cycle.

The core product is written in Java™, and the user interface portion is developed as a Web front-end using JSP™. All the code is centrally located in Austin and is using a standard revision control system. The build process is completely automated with incremental builds occurring daily and full product builds performed every morning at 4 a.m. The current code base is approximately 1.2 million lines of code and is built for deployment on both Microsoft® Windows® and Linux®-based systems.

To improve software quality and reduce long-term security costs, Acme decided to introduce automated source code analysis into their existing software development life cycle. Among the key requirements, Acme requires a testing model that supports their large application as well as the distributed and growing development team. This includes integrating the security analysis into the automated build system to ensure consistency. Frequent reports to management are required to demonstrate progress and return on investment. Ultimately, the team wants to eliminate all high-severity vulnerabilities. The security staff agrees that this can only be accomplished if they can effectively prioritize results, prioritize workflow, and assign remediation to individual developers. The following chart outlines the ability of each model to fill these requirements.

With sufficient resources to achieve their software security goals, Acme makes the decision to implement the centralized model, which meets all key requirements. To perform the vulnerability analysis and prioritize, they create a core security analysis team of release engineers, QA staff, and several developers that have security and application architecture experience. Within the security group, the following roles have been defined:

**Release engineering:** Release engineering is responsible for configuring the analysis tool to be part of the automated build process and incorporating any configuration changes that must be applied as the application is modified throughout development.

**Quality assurance:** The QA team is responsible for initial prioritization. Acme decided that all high-severity vulnerabilities reported during an analysis must be immediately addressed by engineering. The QA team enters new defects for those findings before any other prioritization is performed. The QA team is also responsible for verifying fixes supplied by engineering for any previous defects.

**Security audit:** The security audit team is responsible for creating secure coding policies and customizing assessment rules for the source code analysis. They are responsible for prioritization of any exceptions, questionable vulnerabilities, or design weaknesses of the application. If necessary, the security audit team might add additional defects into the defect tracking system for potential vulnerabilities that need to be addressed.

With the security responsibilities distributed among the different groups within the analysis team, the workflow follows the centralized model closely.

- *A build engineer or release engineer configures the vulnerability analysis to be part of the automated build.*
- *The QA team uses a basic filter applied to the results to view just the high-severity vulnerabilities.*
- *A security team member with application architecture experience prioritizes the rest of the results not filtered out. Ideally after prioritization, every item is categorized and documented so only new items are prioritized in the future.*
- *Developers fix vulnerabilities assigned to them through the defect tracking system. Developers are not required to perform any additional prioritization.*
- *Developers resubmit code with fixes, which are verified by the security analysis team. The cycle begins again, with the release or build engineer possibly needing to reconfigure the analysis to include any additional validation routines or filters for items that were not fixed. This might also be done directly by the engineer.*
- *QA verifies vulnerabilities marked as fixed and opens new bugs for any additional vulnerability. The security team prioritizes new results pointing to potential vulnerabilities.*

**Results**

The separation of responsibilities in this model allows Acme's QA team, security analysts, and developers to concentrate on using their strongest skills. The QA team can quickly identify the high-severity vulnerabilities and assign them as top priority for remediation. The security analysts can focus their efforts on identifying flaws in the application design and creating rules to enforce company policy. The developers do not have to worry about analyzing the results or prioritizing vulnerabilities. The developers can concentrate on making the necessary fixes within the code and checking it back in for the security team to verify. Ultimately, these developers become more adept at secure coding within their area of expertise because of the information provided with the vulnerabilities assigned to them. For example, the developers who interact with the database write better code to prevent SQL injection flaws, but are not experts in session management or cryptography.

The centralized model allows Acme to identify the most critical vulnerabilities and remediate them with maximum efficiency. Conducting the analysis and verification at regular intervals in one location provides consistent reporting on the number of vulnerabilities and remediation progress over time. Within a relatively low number of cycles, the developers demonstrate to managers, partners, and customers, that their source code testing efforts are in fact improving the security of their software and reducing ongoing costs of maintenance, patches, and potential breaches.

## Notes

## Notes

## For more information

To learn more about the IBM Rational® AppScan® Source Edition, please contact your IBM marketing representative or IBM Business Partner, or visit the following Web site: **ibm.com**/software/rational/products/appscan/source/

Ryan Berg is a Senior Security Architect at IBM. Ryan is a popular speaker, instructor, and author in the fields of security, risk management, and secure development processes. He holds patents and has patents pending in multilanguage security assessment, kernel-level security, intermediary security assessment language, and secure remote communication protocols.

[1] "Study: Flaw disclosure hurts software maker's stock," Robert Lemos, SecurityFocus (06/06/05).

[2] "Software Defect Reduction Top 10 List," B. Boehm and V. Basili, IEEE (01/2001).

[3] "Model Description Document," Systems Security Engineering, Capability Maturity Model Version 3.0 (6/15/03) http://www.sse-cmm.org/docs/ssecmmv3final.PDF

***TAKE BACK CONTROL WITH*** **Rational.**

RAW14199-USEN-00