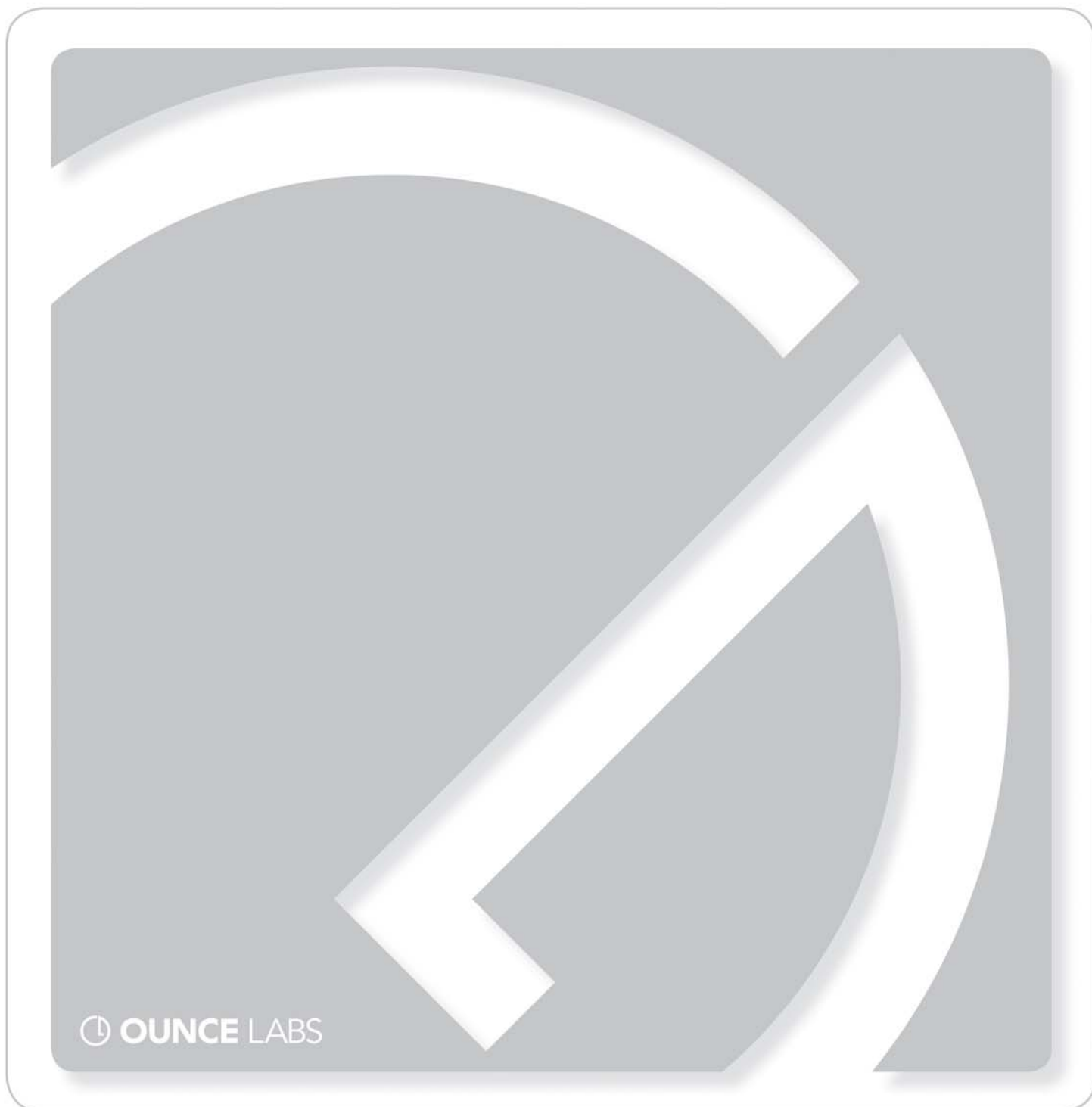


Secure at the Source:

IMPLEMENTING SOURCE CODE VULNERABILITY TESTING
IN THE SOFTWARE DEVELOPMENT LIFE CYCLE



Secure at the Source:

IMPLEMENTING SOURCE CODE VULNERABILITY TESTING IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

By Ryan Berg
Co-Founder and Chief Scientist
Ounce Labs

ABOUT THE AUTHOR

Ryan Berg is a Co-Founder and Chief Scientist for Ounce Labs. In addition to advancing the state of the art in application security technologies, Ryan is also 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 multi-language security assessment, kernel-level security, intermediary security assessment language, and secure remote communication protocols. Prior to Ounce, Ryan co-founded Qiave Technologies, a pioneer in kernel-level security, which later sold to WatchGuard Technologies in October of 2000. In the late 1990's, Ryan also designed and developed the infrastructure for GTE Internetworking/Genuity's appliance-based managed firewall and security services.

Ounce Labs, Inc. is a security solutions company based in Waltham, MA. For more information, please contact us at 781-290-5333.

Doc.#20060706-1.0

Copyright © 2007 by Ounce Labs, Inc. All rights reserved. Ounce Labs, Ounce Security Knowledgebase, and V-Density are trademarks of Ounce Labs, Inc. in the United States and/or other countries. All other trademarks and tradenames are the property of their respective owners.

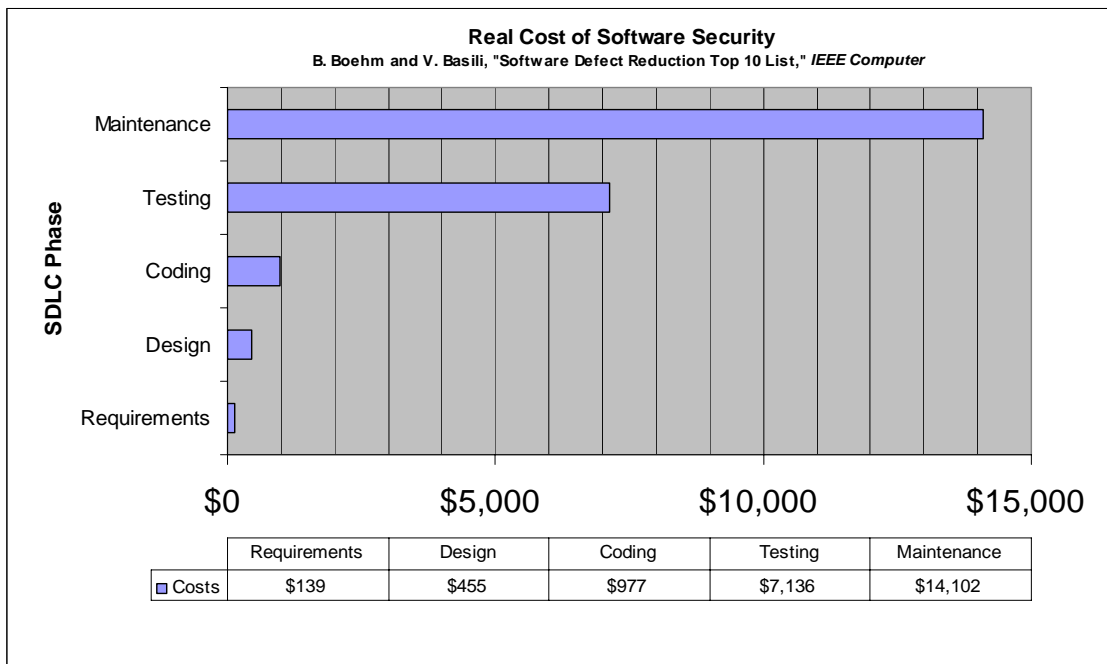
Organizations should implement source code security scanning tools as part of the software development life cycle to find and fix the highest number of security issues early in the project. This will result in a higher-quality product and lower overall application life cycle costs.

Gartner Research¹

Countless studies and analyst recommendations suggest the value of improving security during the software development life cycle (SDLC) 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 even 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.²

Studies with this level of detail are not available for flaws found in custom enterprise software developed in-house or outsourced, but in all cases there is agreement that the earlier in the life cycle that flaws are discovered, the cheaper they are to address. As shown in the chart below, research published by B. Boehm and V. Basili 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.³ For security defects, late-stage costs are often much higher, because in addition to having to remediate the flaws, successful exploits may lead to data theft, sabotage, or other attacks.



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 each vulnerability at the precise line of code and detailing information about the type of

flaw, degree of criticality, and how to fix it. Penetration testing 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 Security In

Among the hurdles that may impede security testing in the SDLC, 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 staff is 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 these two groups, and 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 SDLC process may 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 may 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, and 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, an option that most organizations certainly cannot afford in the long term.

Fiction:

We are already doing peer review; therefore we don't need additional "security" code review.

Fact:

Peer review is not a substitute for security review. Peer review is typically used to find functional bugs, so unless the review is targeted to find security defects and, more importantly, the reviewers have a deep understanding of application security, many of the more critical security vulnerabilities and design flaws will be missed. In many cases the best-intentioned user requirement implemented without functional error can lead to the greatest security risk.

Core Responsibilities

Once past the initial hurdles, 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 below 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. (An excellent guide to creating a process for improving security can be found at <http://www.sse-cmm.org/docs/ssecmmv3final.pdf>.) 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 in this paper address the common elements that should 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 what should be considered 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.

Triage results: Staff members with knowledge of security and of the application study results to prioritize remediation workflow.

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 those that are 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, where software engineers scan their own code after making changes or additions, then remediate any vulnerabilities before checking files back into the source code control system. Ideally, individuals scan the entire application to analyze their code in context so they can also track dataflow and potential flaws across the complex interactions of various files and functions. This may be extremely time-consuming however, unless the code base is relatively small.

How Does it Work?

This model is assumed to be implemented as part of an existing SDLC during the development phase. It requires the developers involved to have sufficient security knowledge not only for identifying vulnerabilities and performing appropriate triage, 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 and/or a fix.

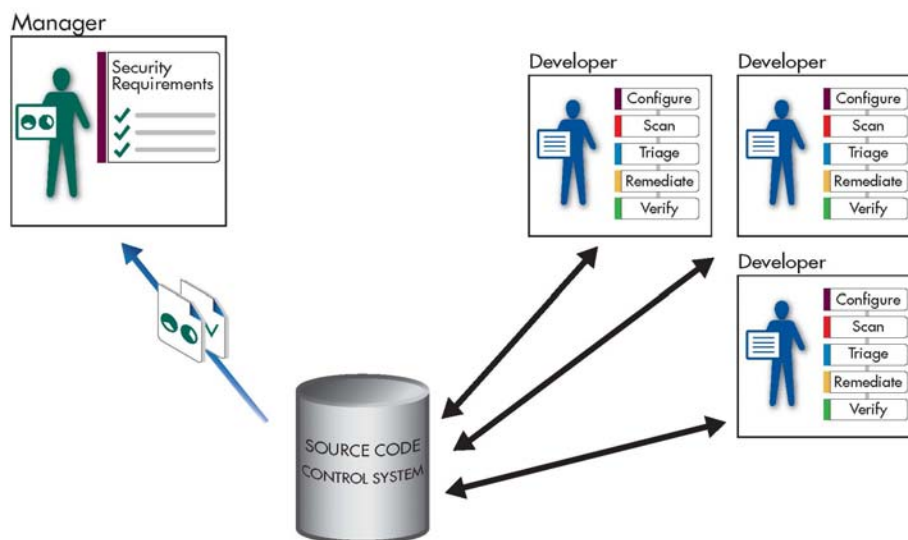
Managers should be able to track development progress to measure delays caused by these security efforts, but generally this model lacks higher-level reporting on vulnerabilities found and fixed, because theoretically the vulnerabilities identified are eliminated before entering into a centralized reporting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

When Does it Work?

Organizations with small development teams and small applications generally 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 will result in effective remediation. Although tracking improvements in this model is difficult, it is expected that the developers will learn from their security efforts and avoid common mistakes as they write future code. Because of these benefits and the relatively little investment required to initiate the Independent Model, it proves to be a feasible method for introducing security into small development organizations. It is also a possible option

for simple system integrator projects, where analysis and remediation is performed by only a few developers.

Independent Model



Workflow:

Managers

- Define security requirements

Developers

- Check-out code
- Add/modify functionality
- Configure scanner
- Scan application

- Triage 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, security-minded developers take a mentorship role in identifying and remediating flaws to help their colleagues better understand the basics of secure development.

When Doesn't it Work?

The Independent Model is not scalable to beyond 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 they do 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 may 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 company-wide (e.g. what level of encryption to use, how to validate input, etc.), 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 will be 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/or train a security-capable member of the development team to act as a mentor, guiding other developers on analysis, triage, 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, although in this case the vulnerability scanning is conducted centrally on the complete application. Typically, the quality assurance 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?

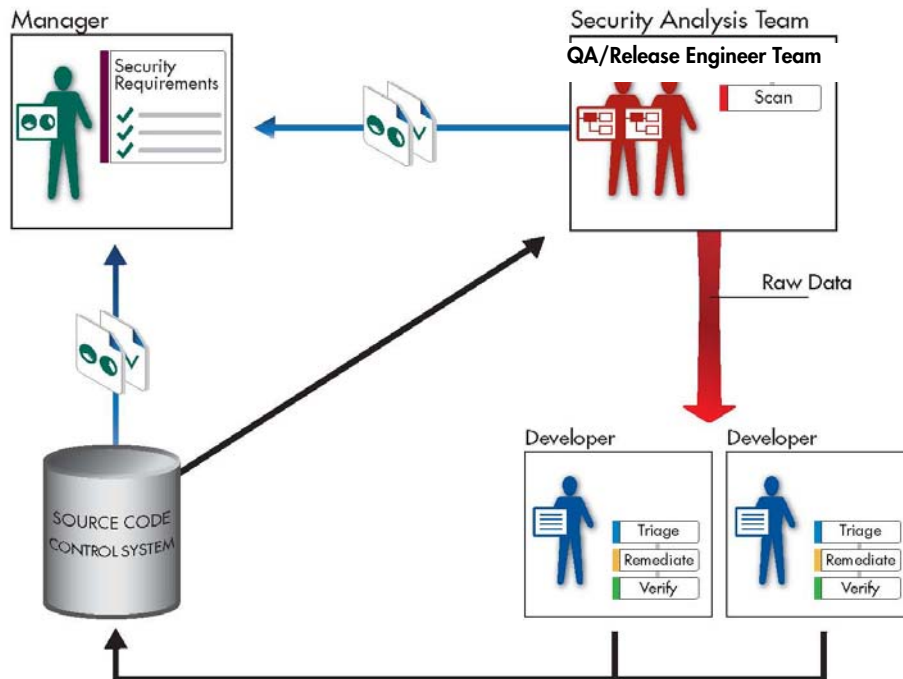
The security analysis can take place either as a requirement before entering the quality assurance phase or as part of an acceptance test after QA has begun. The QA or release engineering team configures the analysis according to centralized security requirements, scans the entire application, and distributes the results as raw data to the development team. Individual developers are then responsible for sifting through 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, while the QA/release engineering team carries out only simple configuration and scheduling tasks.

Depending on the size of application and number of vulnerabilities found in the initial scan, several iterations of security testing may 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, because new dependencies and interactions in the code may 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, because 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.

Distributed Model



Workflow:

Managers

- Define security requirements

QA/Release Engineer team

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

Developers

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

Managers

- Track development progress and re-view assessment data

For cases in which developers do not have sufficient security experience, specialists may be brought into this model to receive results of the analysis and perform the necessary triage. 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 however, since 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 Doesn't it 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 especially in the triage responsibilities for the developers, who 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 triage 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, and maintain a regular, 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, triage, and peer review to verify fixes.

III: Centralized Model

The Centralized Model is the most flexible approach, able to 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, because of the initial investment in resources required. In most cases, source code analysis programs that start with one of the two other models will 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 triaged by this security team as well, so the information they generate provides developers with a prioritized remediation workflow based on the criticality of vulnerabilities. They are also more likely to properly interpret security requirements when they configure the analysis because they understand the business-level issues of risk management.

Ideally, vulnerability remediation assignments are categorized and sent to individual developers through a defect tracking system (DTS) which allows the entire team to monitor progress of flaws being fixed. Developers meanwhile are allowed to focus much more of their time on advancements and improvements to the source code, whether 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 not only context specific, but also incorporates remediation guidelines according to corporate policy. For example, the audit team may find an SQL injection vulnerability in the credit card number processing of an online billing application. The recommended fix could be to modify the SQL to use a stored procedure or parameterized SQL statement, or it could be to use the corporate standard credit card validation routine. The central security team will be able to correctly report this vulnerability and assign remediation based on company-specific policies, instead of requiring developers to make individual decisions.

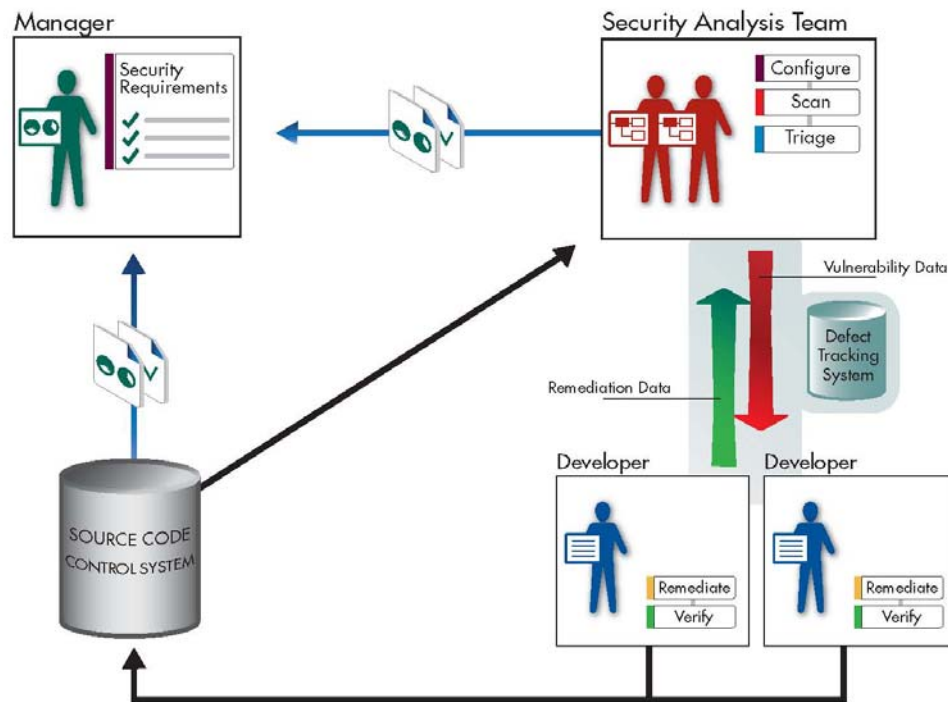
In this model developers may or may not run their own scans to verify that their fixes are effective, but because the scope of a vulnerability may only manifest itself when the entire application is analyzed, it is usually better for the security team to own responsibility for verification.

When Does it Work?

The Centralized Model's primary advantage is the flexibility to integrate efficiently either inside the SDLC, as a complement for internal software audit teams, or externally as a tool for security integrators or code review services. It can be used independent of a formalized development process, though its effectiveness may be diminished without a defined structure. Centralizing the configuration, analysis, and triage makes this the easiest model to manage, and it has the capability

to scale quickly as the development team or project scope increases.

Centralized Model



Workflow:

Managers

- Define security requirements

Security Analysis Team

- Configure scanner for build integration
- Retrieve code for analysis
- Scan entire application
- Triage 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, external code review, etc.), it often requires different delivery methods for the source code. Connecting to the source code control system may work well if the analysis team is inline with the SDLC, but external review teams will most likely request remote access or code delivered on portable storage devices, such as a CD or USB drive.

When Doesn't it Work?

The Centralized Model can be applied to all project scenarios, though it may be more complicated than necessary for small teams or applications. As well, it requires support and commitment from management to assure that sufficient resources are allocated and 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 should not expect a measurable return on investment.

Best Practices

For best results in the Centralized Model:

- Include both security and development in planning and information sharing prior to implementation to achieve agreement on key elements such as:
 - application design and architecture
 - security requirements
 - security training
 - prioritization of remediation workflow
- Provide guidance for the analysis team to assure that their results, triage, and vulnerability assignments achieve the greatest impact.
- Integrate 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 described above 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, so the models may ultimately take on different shapes. However the fundamental stages and functions described above serve as excellent guidance for organizations looking to begin implementing security testing during development.

When choosing one of the three models as a template, it is important to first catalogue existing resources (security expertise, technologies, service partners, etc.) as well project objectives (fewer security patches, competitive advantage, compliance, etc.). Appendix I describes a hypothetical decision-making and implementation process based on factors that real-life 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 much more often, and to a greater degree of success. Companies that are able to reduce vulnerabilities in applications before they are shipped or deployed are recognizing tremendous cost savings both internally and for customers, partners, and other stakeholders that rely on their software.

References:

¹ "Implement Source Code Security Scanning Tools to Improve Application Security," Amrit Williams, Gartner (4/4/06)

² "Study: Flaw disclosure hurts software maker's stock," Robert Lemos, SecurityFocus (06/06/05)

³ "Software Defect Reduction Top 10 List," B. Boehm and V. Basili, IEEE (01/2001)

⁴ "Model Description Document," Systems Security Engineering, Capability Maturity Model Version 3.0 (6/15/03) <http://www.sse.cmm.org/docs/ssccmmv3final.pdf>

REQUIREMENTS	INDEPENDENT MODEL	DISTRIBUTED MODEL	CENTRALIZED MODEL
Effectively reduces security vulnerabilities	✓	✓	✓
Centralized vulnerability and progress reporting		✓	✓
Analysis integrated with automated build system		✓	✓
Supports distributed development teams	✓	✓	✓
Remediation assigned to individual developers			✓
Ability to prioritize workflow by criticality			✓
Scalable for 50 developers and up, as team grows		✓	✓
Supports large applications			✓

Appendix I: Case Study Using the Centralized Model

Consider Acme Software as 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 and QA, located at its company headquarters in Austin TX, and an additional outsourced development team of 25 engineers located in India. The engineering team follows an agile software development process with multiple release and testing phases throughout the SDLC.

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 has been completely automated with incremental builds occurring daily and full product builds performed every morning at 4:00 a.m. The current code base is approximately 1.2 million lines of code and is built for deployment on both Windows and Linux-based systems.

To improve software quality and reduce long-term security costs, Acme has decided to introduce automated source code analysis into their existing software development life cycle. Among key requirements, they require a testing model that can support 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 will be required to demonstrate progress and return on investment, with the ultimate goal of eliminating all high-severity vulnerabilities. The security staff agrees that this can only be accomplished if they can effectively triage all 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 triage, 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 may need to be applied as the application is modified throughout development.

Quality Assurance: The QA team is responsible for initial triage. Acme has decided that all high-severity vulnerabilities reported during an analysis must be immediately addressed by engineering, so QA will enter new defects for those findings before any other triage is performed. QA will also be responsible for verifying fixes supplied by engineering for any previous defects.

Security Audit: This team will be responsible for creating secure coding policies and customizing assessment rules for the source code analysis. They will also be responsible for triage of any exceptions, questionable vulnerabilities, or design weaknesses of the application. If necessary, they may also 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 triages the rest of the results not filtered out in step two. Ideally after triage, every item will be appropriately categorized and documented so only new items will need to be triaged in the future.
- Developers fix vulnerabilities that have been assigned to them through the DTS and are not required to perform any additional triage.
- Developers resubmit code with fixes, which will be 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 and/or filters for items that will not be fixed (this may also be done directly by the engineer)
- QA will verify vulnerabilities marked as fixed and open new bugs for any additional vulnerabilities (the security team will triage 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. Finally, the developers don't have to worry about analyzing the results or triaging vulnerabilities, so they can concentrate on making the necessary fixes within the code and checking it back in for the security team to verify. Ultimately, these developers will 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 will write better code to prevent SQL Injection flaws, but don't need to become 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, they will be able demonstrate to managers – as well as 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.