

realtimepublishers.comtm

Tips and Tricks
Guidetm To

Software Security
Assurance

Klocwork[®]

Kevin Beaver

Introduction to Realtimepublishers

by Sean Daily, Series Editor

The book you are about to enjoy represents an entirely new modality of publishing and a major first in the industry. The founding concept behind Realtimepublishers.com is the idea of providing readers with high-quality books about today's most critical technology topics—at no cost to the reader. Although this feat may sound difficult to achieve, it is made possible through the vision and generosity of a corporate sponsor who agrees to bear the book's production expenses and host the book on its Web site for the benefit of its Web site visitors.

It should be pointed out that the free nature of these publications does not in any way diminish their quality. Without reservation, I can tell you that the book that you're now reading is the equivalent of any similar printed book you might find at your local bookstore—with the notable exception that it won't cost you \$30 to \$80. The Realtimepublishers publishing model also provides other significant benefits. For example, the electronic nature of this book makes activities such as chapter updates and additions or the release of a new edition possible in a far shorter timeframe than is the case with conventional printed books. Because we publish our titles in “real-time”—that is, as chapters are written or revised by the author—you benefit from receiving the information immediately rather than having to wait months or years to receive a complete product.

Finally, I'd like to note that our books are by no means paid advertisements for the sponsor. Realtimepublishers is an independent publishing company and maintains, by written agreement with the sponsor, 100 percent editorial control over the content of our titles. It is my opinion that this system of content delivery not only is of immeasurable value to readers but also will hold a significant place in the future of publishing.

As the founder of Realtimepublishers, my *raison d'être* is to create “dream team” projects—that is, to locate and work only with the industry's leading authors and sponsors, and publish books that help readers do their everyday jobs. To that end, I encourage and welcome your feedback on this or any other book in the Realtimepublishers.com series. If you would like to submit a comment, question, or suggestion, please send an email to feedback@realtimepublishers.com, leave feedback on our Web site at <http://www.realtimepublishers.com>, or call us at 800-509-0532 ext. 110.

Thanks for reading, and enjoy!

Sean Daily
Founder & Series Editor
Realtimepublishers.com, Inc.

Note to Reader: This book presents tips and tricks for four software security topics. For ease of use, the questions and their solutions are divided into topics, and each question is numbered based on the topic, including

- Topic 1: Managing for Software Security
- Topic 2: Developing for Software Security
- Topic 3: Auditing for Software Security
- Topic 4: Testing for Software Security

Introduction to Realtimerepublishers..... i

Topic 1: Managing for Software Security1

Q 1.1: What software security-related problems exist in business today and what are the underlying causes?1

The Root Cause.....1

Contributing Factors2

Q 1.2: What software is affected by poor quality and security vulnerabilities?.....4

Q 1.3: As a software development manager, why should I be worried about reducing the number of vulnerabilities in the software my teams produce?5

Q 1.4: What are some tangible benefits of enhancing the security of our software?6

Q 1.5: What can I do to feel confident answering the common question “How do I know your software is secure?”7

Q 1.6: Is there a solid security strategy I can implement to help ensure my teams build solid applications?8

Integration with the Software Development Life Cycle.....8

Establish Goals.....9

Proven Strategies9

Q 1.7: With so many security technologies at our disposal, how it is possible that software is still being compromised and applications are still being attacked?.....10

Q 1.8: At what point in the development process should we focus our efforts in order to minimize software security vulnerabilities?12

Q 1.9: Which software components and functions are plagued by the most security vulnerabilities and why?13

Q 1.10: As a business executive, why should I be worried about security problems with my company’s software?14

Q 1.11: Are there specific areas I should be concerned with regarding the identification and removal of software security vulnerabilities that arise related to offshoring?.....16

Q 1.12: Are there specific software security areas I should be concerned with or that might require a specific approach related to mergers and acquisitions?.....17

Topic 2: Developing for Software Security18

Q 2.1: Why should software developers be bothered with tacking on security as yet another task to have to worry about?.....18

Q 2.2: What key areas should our development team focus on to ensure the most solid and secure applications long term?19

Q 2.3: What are some commonly overlooked software security vulnerabilities?20

 Revealing Comments20

 Buffer Overflows20

 Mishandling Passwords and Logins.....20

 Assumption that Encrypting Data in Transit Means Everything Is Secure21

 Not Considering the Network or Operating System Layers21

Q 2.4: How can software security vulnerabilities be categorized so they’re easier to understand?22

 Denial of Service.....22

 Authentication Weaknesses22

 Input Attacks.....23

 Directory Traversals.....23

 Improper Storage of Files and Data.....23

Q 2.5: Are there any common software development practices that stand out as serious risks?...24

Q 2.6: Are there other technologies or layered security measures we can integrate into our software to help prevent various attacks?24

Q 2.7: What role does a layered security defense play in software development?.....25

Q 2.8: There is a general consensus in my development lab that as long as firewalls and Secure Sockets Layer (SSL) are used, the application is secure—is this true?27

Q 2.9: Do I have anything to worry about as long as I develop software with the Open Web Application Security Project (OWASP) Top 10 vulnerabilities in mind?.....27

Q 2.10: What development practices can we integrate into our daily development routines to reduce the number of security vulnerabilities?29

Q 2.11: What development tools can we integrate into our daily development routines to reduce the number of security vulnerabilities?.....30

Q 2.12: Is it really worth encrypting our database if we have implemented various security layers in our software? If so, what parts of the database should we encrypt?.....30

Topic 3: Auditing for Software Security32

Q 3.1: What benefits will a formal software audit offer compared with other types of software security testing?32

Q 3.2: What are the different types of tests and tools I can use to assess the security of my software?.....32

Q 3.3: How can I determine whether we should perform a software security audit?.....33

Q 3.4: Who should perform formal software security audits?.....33

Q 3.5: Which areas of our software should we audit?34

Q 3.6: What are some common software security auditing mistakes?35

Q 3.7: How can I determine whether we need a formal software security audit or other type of test such as a manual code review or penetration test?.....35

Q 3.8: What features should I look for in software security testing tools?.....36

Q 3.9: Would a software security audit performed by an external and independent consultant produce better results than one performed by our own internal IT auditing staff?.....37

Q 3.10: Are there specific software security standards and best practices we can look for to ensure we’re getting the most from our auditing investment?.....38

Q 3.11: Is a source code audit all that is needed to identify the big picture software weaknesses as well as granular vulnerabilities?38

Q 3.12: What should we do once security vulnerabilities have been identified by an audit?39

Topic 4: Testing for Software Security.....40

Q 4.1: What methods are available for my developers and quality assurance staff to do their own software security testing and what are the differences between each method?40

Q 4.2: Who should perform software security testing?40

Q 4.3: What software security testing techniques should my quality assurance engineer be utilizing?41

Q 4.4: What software security tests should be performed during a penetration test?.....41

Q 4.5: What are the benefits of performing static analysis code reviews compared with runtime penetration testing?42

Q 4.6: What steps should be taken to ensure the security vulnerabilities found during the testing phase are properly addressed?42

Q 4.7: What is the difference between traditional testing and testing for security?.....43

Q 4.8: Are there any testing tool features that should exist above and beyond those needed to perform higher-level audits?.....43

Q 4.9: Is it better to have an external and independent expert perform software security testing?44

Q 4.10: What software security testing tools are the best fit for quality assurance (QA) professionals?.....45

Q 4.11: How are penetration testing tools different from typical software security testing tools?46

Q 4.12: Should we focus time, money, and efforts on performing code reviews, penetration tests, or both?47

Q 4.13: What are some common oversights and mistakes made once the software testing phase is complete?48

Copyright Statement

© 2005 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at info@realtimepublishers.com.

Topic 1: Managing for Software Security

Q 1.1: What software security-related problems exist in business today and what are the underlying causes?

A: Software security has recently stolen the spotlight from the typical information security focus on viruses, firewalls, insecure network protocols, and the like. Although most of the information threats remain the same, IT industry professionals agree that insecure software is one of the root causes of security breaches and often leads to a number of business problems. In fact, poor software security is arguably as big of an issue as the often-cited insider threat that exists in computer networks—the untrained, careless, and malicious employees that, both intentionally and unintentionally, wreak havoc on information systems.

At a high-level, software security vulnerabilities adversely affect all types of businesses, from independent software vendors (ISVs) who develop and sell their own software products to large corporations that purchase software products for everyday business tasks and practically every type of user in between. The negative impact of software security problems is seemingly unending. ISVs look bad and can lose business when their software is found to be vulnerable to attack, and businesses and individuals alike must deal with system unavailability as well as having their sensitive information being compromised. These outcomes not only cause frustration but can easily lead to liability issues, identity theft, and even failure to meet regulatory compliance requirements.

The Root Cause

The root of the software security problem can be reduced down to the fact that development managers and security managers are coming at this from two different ends of the business spectrum. The same goes for hands-on developers and information security assessment experts. Simply put, these roles have different sets of goals. On the development side, the main goal is to deliver solid software containing the functionality requested within a specific timeframe. On the security side, the main goal is to assess risks and find solutions for the critical issues as soon as possible. These goals are in direct conflict with one another, especially after you add in the demands and pressures coming down on developers from the heads of marketing and sales, and the security requirements coming down on security administrators and auditors from the heads of IT and compliance.

Software developers, and even certain software development managers, are often too familiar with their own code and processes to be able to see the big picture security problems. Security managers and network administrators often react to software security problems by implementing more software and hardware—often expensive products in the form of network and application firewalls, intrusion prevention systems (IPSs), virtual private networks (VPNs), and strong authentication systems.

In fact, the pervasiveness of software vulnerabilities has created a multi-billion dollar industry of security add-ons and multi-billion dollars in losses due to software exploits and patch management needs. There is too much reliance placed on these security products, which creates a false sense of security among administrators and especially upper managers. These products are not the answer to software security problems but rather patches to place over fundamental security flaws at a much lower level that (ideally) shouldn't be there in the first place.

Security technologies have their place in creating a layered defense, but it's becoming a well-known fact that security add-ons cannot be relied upon exclusively. For example, firewalls must allow certain traffic into the network thus allowing an attacker to sneak in, and encryption often only serves to hide the fact that an information security breach occurred.

Contributing Factors

It's easy to see that there is a major business problem with no simple solution. Looking at the software security problem more closely, there are several additional contributing factors faced in business today:

- By and large, developers don't understand security—at least to the level at which they should. In fact, most are in over their heads simply trying to accomplish the basic feature sets that were promised to be delivered. However, there are few information security professionals that truly understand software development concepts. Many information security professionals don't understand programming and development to the levels at which they need to in order to understand the root causes of the very vulnerabilities for which they're searching.
- Many software developers and development managers don't understand the risks and business impact of insecure software. In addition, they're not looking at the big picture and long-term consequences. Such is especially true with niche programmers coding small or standalone pieces of an application as opposed to developers working at a higher level who understand the bigger picture. The paradigm must shift to include an understanding of the business ramifications associated with increasingly complex software that runs on networked and mobile computer systems.
- The past paradigm of software security is that safeguards and controls should be implemented by the operating system (OS) or at the network level. As demonstrated by the common weaknesses associated with firewalls and “allow anyone to do anything” default OS configurations, such safeguards cannot be relied on exclusively.
- Architectural issues with highly vulnerable legacy software (for example, Microsoft Windows NT) prevent the remediation of many security vulnerabilities without having to perform a serious overhaul or even completely re-write underlying code. In addition, there are complexity issues and inherent vulnerabilities with modern languages such as Java, C, and C#.

- Secure software is not usually the result of fast or efficient coding. It takes time, but in the business world, time is money. Time-to-market pressures from marketing, sales, and other powers-that-be plays a large role in software security problems. The focus has been more on features, getting it done, and getting it out the door more than anything else. This occurrence is a side-effect of lack of security awareness and placing bottom-line numbers first—both of which can be overcome but not without the adjustment of priorities and proper buy-in from key decision-makers.
- The core elements of formal security policies and security standards are often absent—a major development mistake. Any business function needs structure, goals, and practices reflecting “here’s how we do it here” in order to be effective. Such is especially true when it comes to integrating security into the software development process.
- Attackers are quicker to the draw and often have a lot of anonymous time to work on their exploits. In addition, their security skills are often more advanced than those of software developers. These shortcomings help fuel the reactive hack-and-patch syndrome that business applications are subjected to today.
- Customers haven’t demanded more secure software until recently, giving developers, development managers, and business decision-makers less of an incentive to improve on the issue.
- Not only are software defects difficult to detect (especially in software that is working fine otherwise), but the attacks and malware exploiting these defects often go unnoticed as well.
- The complexity of information systems is growing exponentially. Software is rarely run in a standalone fashion. Instead, even at the highest level, there are often myriad system dependencies including hardware, application servers, back-end databases, Internet connectivity, and more.
- Software is becoming increasingly complex and extensible especially with applications supporting ‘mobile code’ built around .NET and Java. Most modern OSs such as Linux and Windows XP and even some office applications such as Microsoft Office are comprised of millions of lines of code. Even common internal business applications are made up of hundreds of thousands of lines of code. This complexity introduces bugs and security flaws.
- Poor software development habits including the lack of a formal methodology, developing software in non-secure environments, and not using software metrics to enhance the process of measuring and controlling defects.

A common argument is that the software industry is in its infancy and is simply suffering growing pains. This reality may be true for certain developers and development languages, but by and large, most of these underlying software security problems have been around for decades. Case in point is the Morris Worm that exploited a buffer overflow in the UNIX finger service way back in 1988! The bottom line is that security vulnerabilities are increasing and software defects are not decreasing—two key ingredients in the recipe for information systems disaster.

Q 1.2: What software is affected by poor quality and security vulnerabilities?

A: Businesses and individuals alike have come to completely rely on software to deliver the right functionality at any given time. Software is pervasive throughout businesses and even affects most individuals in very personal ways. Software is embedded in practically everything we use—from cell phones and automobiles to network routers and back-end database systems. Every industry from manufacturing to healthcare uses software to accomplish business tasks. Poor software quality and system downtime related to security exploits affect many people in many different ways.

All types of software—from Web browsers to embedded HVAC control systems to highly-complex e-commerce applications—are impacted by poor quality and security vulnerabilities. Regardless of the code medium—firmware, standalone programs, operating systems (OSs), and so on—any type of bug can introduce stability and reliability problems. This situation, in turn, affects the cornerstones of solid information security, creating problems with confidentiality, integrity, and system availability—all areas that businesses cannot afford to have compromised.

A critical part of understanding software security is to define security vulnerabilities and determine when a vulnerability, is indeed, a vulnerability. A security vulnerability is defined as a flaw or weakness (in this case a software bug) that can be exploited by a threat (an attacker or malware) to cause harm or damage. However, not every software bug is a security vulnerability, so what constitutes a true security vulnerability? Technically, a vulnerability that can be relatively easy to exploit and

- Cause software to hang or crash (considered a Denial of Service—DoS—attack)
- Allow for privilege escalation on the system (for malware or an attacker to be promoted from a standard user to an administrator-level user in order to do more damage)
- Allow for execution of “arbitrary code” that can execute specific instructions, install backdoors, steal information, and more
- Cause software to act in unintended ways

The critical issue to remember is that no software is immune from poor quality or security vulnerabilities. No matter which development language or OS platform is used, software flaws can crop up anytime and anywhere.

Q 1.3: As a software development manager, why should I be worried about reducing the number of vulnerabilities in the software my teams produce?

A: There are several underlying business-related reasons why reducing software vulnerabilities is important. First and foremost, it costs money. Fixing software security defects after the fact is much more expensive than doing it right the first time. This has a great impact on development budgets and resources.

Looking deeper, the software development function can be generically compared with a business—there are customers driving certain demands and people work together to meet those demands. There are both long-term strategies and short-term tactics that must be executed at the right place and right time to keep customers happy and wanting to come back for more.

Take the current software quality problems and juxtapose them into another industry such as automobile manufacturing to illustrate the possible negative outcomes of poor development quality. In fact, if a major automobile company followed the path of poorly written software and put out unsafe vehicles that are susceptible to known failures at any given time, can you imagine the uproar, the injuries, and the lawsuits? All in the name of ‘time-to-market’ or ‘features first.’

Imagine what would happen to development managers whose teams generate sloppy code if they were out on their own trying to make such a ‘business’ prosper. Many development teams have gotten away with it, but history tells us that the market likely won’t have it that way for long. We’re arguably at that point in time where those who don’t integrate secure coding practices into their development processes and generate higher-quality software won’t survive in the marketplace.

Vulnerable software contributes to a multi-billion dollar problem, based solely on costs associated with reported attacker and malware attack losses. There are untold software security problems and exploits (and thus costs) that go undiscovered and unreported. Two rather obvious negative side-effects of software vulnerabilities are unhappy management and let-down customers. However, there are various indirect business-related consequences that must be considered as well, such as a reduction in customer referrals and even cash flow problems caused by customers who don’t want to pay for defective goods. Moreover, several technical issues result from insecure software such as:

- Data corruption
- Rogue commands executed
- Unnecessary system resource usage
- Denying service to legitimate system requests
- Entire networks brought down
- Installation and propagation of malware—including rootkits—throughout the network

In addition, poor development practices can lead to numerous ongoing software patches and hotfix requirements. The obvious benefit is that the issues are remediated and the software is perceived to be more solid. However, the ongoing pattern of break-fix also generates several side-effects for both the developer and the end user including:

- Increased number of support calls
- Having to support customers running code that is several revisions old
- Greater odds of failing to meet support service level agreements (SLAs)
- Increased systems administration
- Potential for regression (the introduction of new software problems when fixing others)

Odds are, these aren't outcomes that any reasonable development manager would wish on anyone, but they're realistic issues that can't be overlooked nonetheless.

Q 1.4: What are some tangible benefits of enhancing the security of our software?

A: There are many tradeoffs associated with software security. It boils down to increased time and costs on the front end doing the actual development or increased time and costs on the back end trying to keep the software secure by other means. If a good balance can be found, the list of benefits associated with enhanced software security is limitless including:

- Customer-related benefits
 - Customer recognition that the organization takes security seriously
 - Increased customer satisfaction and loyalty
 - Increased customer referrals
 - Enhanced value and return on their investment
- Development-related benefits
 - Simpler security integration long-term when security is built-in at the earliest possible phase
 - Security problems are much easier to find early on in the initial software design and development phases, minimizing complexities associated with fixing the problems and reducing development-related risks
- Business-related benefits
 - Enhanced industry positioning and reputation
 - Positive revenue and other financial benefits related to selling more licenses
 - Enhanced employee morale knowing that software the business develops, sells, supports, or uses is solid and dependable
 - Improved long-term business viability

- Compliance and legal-related benefits
 - Better opportunities to maintain regulatory compliance internally or help others achieve regulatory compliance by using more secure software
 - Improved ability to meet deliverables, contracts, and associated business responsibilities which can, in turn, decrease legal liabilities
 - Increased protection of intellectual property, individually identifiable information, and other sensitive data

Software quality is directly related to software security, which can have far-reaching and long-term business impacts if achieved. When software is developed with security in mind and the end product is more secure and stable, everyone associated with the software ultimately benefits.

Q 1.5: What can I do to feel confident answering the common question “How do I know your software is secure?”

A: With the myriad privacy and security demands coming from customers, business partners, and regulatory bodies, this question is being asked more and more in today’s business environment. These demands are changing the way organizations think about software quality and security—especially in the area of product marketing. Many in-house developers and independent software vendors (ISVs) claim that their software is secure—some, such as Oracle, go as far as to claim that their software is unbreakable.

Like most reality veiled by marketing hype, more often than not, software isn’t as secure as it’s claimed to be. This reality sets up everyone involved for failure. If ‘robust’ or ‘secure’ software is claimed, then it must be able to live up to this claim. Those using the software need to know that secure practices are being considered and properly executed. But how can you effectively convey that message?

The best, yet most elusive, method for making your software secure is to integrate solid security practices into the development process. It won’t ensure security, but odds of improvements are in your favor—and it can give you (and other stakeholders) a boost of confidence that things are being done right.

Another effective way to feel strongly that your software is secure is to actually test it using manual or automated code reviews and various other security assessment tools. This step may be enough, but if you dare, you might want to share the results of your security testing with your customers to demonstrate that your secure processes are actually working.

Your software security testing could come in the form of internal tests or you could hire a third-party to look for vulnerabilities from a fresh and unbiased perspective. Both are effective means of discovering vulnerabilities and can be an effective way to prove that your software is secure as claimed.

Your software development practices may also be part of obtaining certain organizational certifications such as ISO 9000-9001 and ISO/IEC 12207 as well as meeting various NIST standards such as Special Publications 800-27, 800-55, and 800-64.

Keep in mind that even though security testing and remediation has taken place or international standards have been met or certifications have been obtained, there is absolutely no guarantee that every possible vulnerability has been discovered and/or prevented. Software and the interconnected information systems it runs on are simply too complex to guarantee that nothing has been overlooked. However, implementing such practices and performing due diligence to do what is right will put your organization well ahead of others in this area—undoubtedly a commendable way of doing business.

Q 1.6: Is there a solid security strategy I can implement to help ensure my teams build solid applications?

A: In an ideal world, software would change very little, if ever. This reality would help ensure software security, but at what cost? Not innovating or responding to market and customer demands is arguably the fastest way for software vendors and developers to crash and burn. Further complicating matters, human involvement in any business process introduces serious complexities that require awareness, training, structure, and effective management.

Instead of pointing fingers and placing individual blame, it makes the most sense to look at this situation from a neutral business perspective. Solid software security involves various factors and various people—a true business problem that requires a solid business solution.

Integration with the Software Development Life Cycle

A fundamental weakness contributing to poor software security is that most security weaknesses are discovered too late in the game. This fact is driving the need for improved development processes and improved methods of testing for security vulnerabilities. In order to build secure software, information security needs to be integrated into every phase of the software development life cycle (SDLC):

- Plan—Determine big picture security goals to answer the why, when, where, and how questions
- Specify requirements—Outline specific controls needed to accomplish security goals
- Analyze—Determine which security features will make it into the final code
- Design—Map out how security controls will be integrated into the project
- Develop—Write the actual security code and/or integrate external security modules and perform static analysis
- Test—Assess the security controls via unit testing (how individual components work) and integration testing (how everything works as a whole)
- Implement—Deploy the software in the specific environment to meet your or your customer's specific needs
- Post-implementation review—Follow up the implementation by retesting the source code and running software
- Maintain—Perform ongoing tests for defect tracking and subsequent improvements

Obviously, all these practices cannot be implemented directly and immediately into existing applications, but they can be integrated over time during major revisions and with new projects.

Establish Goals

A common mistake is to outline security goals during the planning phase—such as confidentiality and integrity of all data and validation of all application input—but then overlook them or place them on the back burner. This happens often by even the most well-intentioned developers. The truly effective software development team needs to have clearly defined steps that outline specifically how each goal will be met. For example, in order to meet a goal of data confidentiality and integrity, the development team must know about and be held accountable for items such as:

- Using TLS (or SSL v3) for all network transactions containing credit card information
- Considering the key length and relative strength of the cipher used
- Encrypting all database tables containing personal healthcare information
- Creating detailed audit log entries each time a user accesses information classified as sensitive or confidential via the application

This process may require the input and oversight of a security-savvy developer or even a technical member of the organization's information security team.

Proven Strategies

Other strategies for ensuring secure applications include:

- Integrating secure development into job description requirements and holding developers accountable
- Providing developers ongoing training and awareness in security concepts and techniques such as what is provided at security conferences, in formal classroom training, and the wide selection of recent books on developing secure software
- Ensuring the lines of communication between developers and security team member is open and utilized
- Focusing on being proactive in addressing known security vulnerabilities—especially in the pre-deployment phase
- Determining, beforehand, how you want to focus on security vulnerabilities once they're identified. This could be:
 - Focusing on issues that can be addressed in the existing development stage
 - Focusing on implementing changes moving forward with any new code
 - Waiting until time permits to go back and fix the problems from the beginning
- Documenting secure software development policies and standards
- Performing static analysis during the development and post-implementation phases

- Integrating well-known software security-related standards and frameworks such as the Open Web Application Security Project (OWASP), the ISO/IEC 17799:2005 Information technology—Security techniques—code of practice for information security management, and ISO/IEC 12207:1995 Information technology—Software life cycle processes
- Establishing specific metrics for software security and measuring for success
- Making quality assurance (QA) a key area of the development environment with the proper resources to affect change
- Continually testing software for new exploits

It's also critical for developers and development managers to be able to assess which issues need priority attention. Vulnerabilities that are known to be easily exploitable and expose sensitive information or harm critical systems should get the most resources and attention.

A buffer overflow that allows for remote code execution is a popular example of a software vulnerability that needs to be given top priority. However, potential weaknesses that may be exploited or pose a threat down the road can be addressed as soon it makes business sense. An authentication algorithm that could possibly be cracked given enough time with the right tools would be an example of a low-priority software vulnerability.

Eliminating all software vulnerabilities and defects found is not realistic. Instead, go for the highest payoff tasks that will have the greatest impact on the overall security and quality of your software.

Q 1.7: With so many security technologies at our disposal, how it is possible that software is still being compromised and applications are still being attacked?

A: At the root of this problem is a disproportionate focus and dependency placed on security technologies by network administrators, IT managers, and even C-level executives. The marketing machine of security safeguard vendors is extremely powerful and influential, pushing security add-ons such as:

- Firewalls
- Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs)
- Strong authentication systems
- Antivirus and anti-spyware software
- Virtual private networks (VPNs) and related data transmission protection measures

According to Gartner estimates in 2004, the majority of the \$5 billion IT security market was spent on these types of security solutions. However, in the real world, it's being proven over and over that these technical solutions are not the answer to software security problems:

- Firewalls can filter traffic and help prevent Denial of Service (DoS) attacks but won't protect software inside the network and only offer very limited protection for applications that must be accessible through the firewall.
- IPSs can generate alerts and sometimes prevent network and host-based attacks but offer little to no protection against application input attacks, application authentication compromises, exception handling, insecure storage or transmission of sensitive data, or malicious activity by anyone with authority on the network.
- Strong authentication systems can verify a user is who the user says he or she is but offer no protection against any unauthorized usage or other software attacks.
- Malware protection software detects malicious signatures and some anomalous behavior but not nearly enough to prevent the seemingly benign software attacks that occur most often.

Most of these technologies *respond* to security breaches rather than *prevent* them, and even when used in conjunction, they cannot provide complete security—not even close.

As with many other deep-rooted business issues, vendor-supplied 'cures' abound while the root causes of the problems are overlooked. The fundamental problem with software security is within the software itself. Subtle (and not so subtle) flaws are introduced, discovered, and exploited over time, which can lead to major business problems for both the software developers and the software users.

The underlying problem is essentially a failure to integrate secure programming into the software development life cycle. This shortcoming encompasses issues such as developers not understanding security, failure to test for security holes via static analysis and penetration testing, and management placing more value on functionality and hard deadlines than security.

Developers and systems administrators are also failing to look at their software and systems from an attacker's point of view. They're performing their own security reviews but not looking at the software as a whole from different perspectives with different testing tools to determine which parts provide interesting information and which can be exploited.

Technical schools, colleges, and universities have played a significant role in the software security dilemma. In programming courses, the majority of the focus is placed on learning specific languages and programming—instruction sets, syntax, and so forth—rather than learning how to develop larger systems with security in mind from the ground up. This lack of security education from the start further perpetuates the software security problem.

This movement is starting to change, but academia is still far behind what is needed in the industry as evidenced by computer science graduates and entry-level programmers. In fact, simple scans of course offerings at some of the top technical schools in the United States highlight the minimal focus on software security assurance. There are tons of courses offered on software engineering, data structures, and algorithm analysis but only minimal special topics courses dedicated to secure software development.

The bottom line is that more people in marketing and upper management are more concerned with delivering applications to meet every user's needs and desires. There is still a strong belief that security can be added on later. This philosophy helps companies meet quarterly numbers but is very dangerous and shortsighted and ends up taking away from the user experience and creating unnecessary business liabilities long-term.

Q 1.8: At what point in the development process should we focus our efforts in order to minimize software security vulnerabilities?

A: It has been said by software security experts for years that secure coding practices vulnerability testing must be integrated into the entire software development life cycle as Figure 1.1 shows.

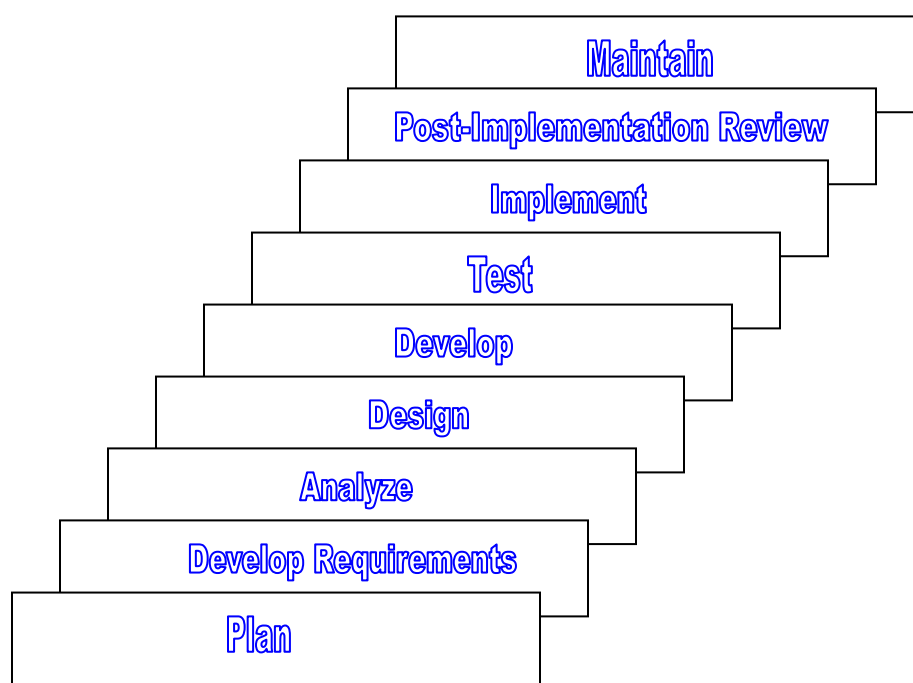


Figure 1.1: The steps involved in the software development life cycle.

Implementing this process is much easier said than done but it's a proven way to develop solid code and prevent unnecessary security flaws. It's therefore critical to ensure that software security goals and metrics are established, buy-in is obtained, and everyone involved with secure development practices (programmers, developers, testers, quality assurance—QA—staff, information security team, and management) is working toward making improvements.

There will likely be existing applications in development and legacy applications that simply cannot be overhauled immediately. However, as a development or application maintenance manager, you can establish mandates based on business needs that can help make immediate improvement such as:

- All new code moving forward will adhere to specific security standards and be measured according to the metrics you've established
- No further development or maintenance occurs unless and until security flaws labeled as critical are addressed

Obviously, risks, rewards, and costs must be balanced when determining how to address current problems. It's therefore essential to include other stakeholders and decision-makers outside of development in the process.

By integrating secure coding practices and security testing into the entire development process, you can help prevent the spread of software security flaws (both known and unknown) when code is reused by developers. In addition, it's much less expensive, less disruptive, and easier to make changes when software security flaws are prevented, or at least tested for, early on in the software development life cycle. If secure development practices are not integrated into the process and security testing is not performed until the latter phases of the development life cycle, software fixes and redesigns are, at best, very difficult to deal with and remediate.

Q 1.9: Which software components and functions are plagued by the most security vulnerabilities and why?

A: Security flaws are introduced to practically every area at every level of the software development life cycle—especially in the *plan* through *develop* steps (for an illustration of the software development life cycle, see Figure 1.1 in Q 1.8). This reality applies to not only commercial software but also in-house applications regardless of the platform or programming language used.

Many security vulnerabilities start at the ground level when development teams overlook security when architecting an application. This shortcoming can come in the form of weak authentication methods, easily cracked encryption cipher suites, or even relying on the underlying operating system (OS) to be secure. As developers get deeper into a project, security vulnerabilities can be introduced by using functions that are known to be vulnerable, such as the infamous C-language `gets()` and `strcpy()`, or through general sloppiness and laziness such as not validating user input into the program.

Generally speaking, memory problems such as buffer overflows, improper format string handling, and integer overflows comprise the majority of software security problems. Other common problem areas include weak password management, flawed access controls, and improper error handling.

In addition, many legacy applications developed before security was an issue and previously shielded from attack inside the corporate network, are being made Internet-accessible. This development introduces a slew of previously undiagnosed flaws for any attacker to exploit.

Interestingly, new software problems are few and far between. Developers are simply introducing the same vulnerabilities and performing the same mistakes over and over again. The vulnerabilities listed in the National Vulnerability Database (<http://nvd.nist.gov>) and Common Vulnerabilities and Exposures (CVE) dictionary (<http://cve.mitre.org/cve>) help prove this.

Q 1.10: As a business executive, why should I be worried about security problems with my company's software?

A: Businesses are working to meet consumer and business partner demands by developing more complex software to facilitate online services, manage supply chains, and more. Unfortunately, more focus is often placed on functionality and deadlines than on creating secure applications. With the growing demand and dependence on complex, interconnected software, there is more to lose now than ever. There is more sensitive personal information being gathered and stored electronically, more industry and government regulatory compliance requirements, and more business dependence on software and computer systems. The fact of the matter is that when software breaks, time is wasted and money is lost.

In a 2004 press release, Gartner stated that enterprise configuration management and security incident response costs would each be reduced by 75 percent if half of all software security vulnerabilities were removed prior to production. Executive management should be concerned about the security of software they're developing and/or using for the following reasons:

- Costs associated with fixing software defects after the fact are much greater compared with fixing them during the development process. Johanna Rothman's study (available at <http://www.jrothman.com/Papers/Costtofixdefect.html>) on this subject is very enlightening.
- The ability to show that security and privacy are taken seriously is being considered by customers, business partners, and regulators.
- Secure software is an effective way to create product differentiation.
- Perceived product value can add to the bottom line.
- Customer satisfaction will be improved through secure software, which can lead to more referrals.
- Secure software ensures repeat business, which can lead to increased revenues.
- Meeting industry and governmental regulatory compliance requirements—such as those imposed by the Gramm-Leach-Bliley Act (GLBA), the Federal Financial Institutions Examination Council (FFIEC), the Health Insurance Portability and Accountability Act (HIPAA), and the Sarbanes-Oxley (SOX) Act—is not optional.
- Secure software provides protection against downstream liability issues.

Organizations are being questioned by customers and business partners about how secure their software is. As a result, and due to legal requirements and increased awareness of the problem, smart business leaders are buying into the need for secure solutions. It would behoove any executive to understand what is taking place and be able to answer this question when prompted.

There are numerous benefits associated with producing secure software that are difficult if not impossible to quantify. Because of the lack of hard numbers to support such claims, these benefits are overlooked but need to be considered nonetheless. When software is secure and stable, the following can occur:

- Increased customer confidence and loyalty
- Increased brand image and perceived product value
- Decreased level of perceived risks when using the software
- Increased stakeholder and shareholder value
- Increased trust in the organization's leadership to do the right thing
- Increased employee morale
- Increased goodwill to customers, the industry, and even the environment
- Improved long-term business viability
- Increased productivity among developers, administrators, and end users

With millions of lines of abstract code, software is extremely complex. This complexity can introduce numerous software defects that can lead to security vulnerabilities—on the scale of 5 to 10 unique flaws per 1000 lines of code! By making software security a top priority, organizations can differentiate themselves from others by offering software that has been validated to be as secure as possible. Solid and secure software can bring peace of mind and will become a point of reference for a growing number of software buyers and users in the future.

Furthermore, the type of software used or the industry your company is in is irrelevant as security problems affect software across the board (albeit the consequences of security flaws are greater for some than others—such as the Department of Defense compared with an online greeting card company). Changing programming languages and operating system (OS) platforms won't make a difference. Hiring new developers and development managers won't likely help either.

What will make a difference are managers and executives who support secure development principles such as sponsoring continuous developer training, establishing and measuring against software security metrics, and encouraging ongoing security testing. It's essential for managers and executives to lead by example and help establish a culture of security—if they don't, then who will?

Q 1.11: Are there specific areas I should be concerned with regarding the identification and removal of software security vulnerabilities that arise related to offshoring?

A: Oftentimes, code is used from third parties without a single thought going into whether the code is secure. When offshoring software development (or any type of development outsourcing for that matter), various precautions need to be taken to ensure not only the protection of source code but also the prevention of security flaws, backdoors, and future liabilities.

Attempting to achieve full control over remote development teams can be difficult, but there are several areas you must focus on to keep from getting burned. Ensure the following questions can be confidently answered to help prevent, detect, and correct problems and risks associated with offshoring and outsourcing:

- Does the contract spell out audit provisions and specific details regarding who is responsible for security defects?
- What constitutes a security vulnerability (buffer overflows, code/command injection, lack of input filtering, poor error/exception handling, and so on)?
- Does the outsourced team understand your security goals and standards?
- Does the outsourced team understand your coding policies?
- Is the security of outsourced code improving, remaining the same, or getting worse?
- Which team or team members are writing code that is not secure? How can this problem be addressed?
- Is someone on your team performing static analysis and penetration testing on new code developed by your outsourcing provider?
- Will an independent outsider be hired to perform an unbiased software security assessment?
- Will details on the security functionality be part of the final deliverables?
- Is a delta analysis being performed to determine the differences (and consequential flaws) between what you sent out compared with what you get back?

Establishing security criteria, ensuring that outsourced teams understand your security needs, and holding outsourced teams responsible for software defects is not a simple task but needs to be part of the overall process. Including a code integrity warranty such as those dating back to the original mainframe days may seem like overkill and may even be impossible to put in place but is worth considering. By getting the legal department, information security, and other key decision makers involved in this process, you'll know you've performed your due diligence and everyone will go into the business venture with their eyes wide open.

Q 1.12: Are there specific software security areas I should be concerned with or that might require a specific approach related to mergers and acquisitions?

A: Mergers and acquisitions are a complex topic that requires experienced legal and business advice. As with offshoring and outsourced software development, it's critical to ensure everyone is on the same page going into business ventures such as these.

It's not uncommon for developers and development managers to have complex software that is riddled with security holes dumped onto their plates as part of a business merger or acquisition. Key areas of due diligence to ensure software security assurance isn't pushed to the wayside during mergers and acquisitions include:

- How do you know what is contained in the merging/acquired company's source code? Will they readily share their source code and perform the appropriate knowledge transfer?
- Is the merger or acquisition contingent upon an internal or third-party review of existing source code?
- What tests have been and will be run to understand inherent software security risks?
- Who is responsible for cleaning up software riddled with security flaws?

These business concerns may not fit politically or be addressable financially during a merger or acquisition. It's the job of software development managers to at least ensure key decision makers are made aware of these issues and make the final call to move forward.

Topic 2: Developing for Software Security

Q 2.1: Why should software developers be bothered with tacking on security as yet another task to have to worry about?

A: It's true that marketing, upper management, and customers are demanding more from developers—such as complex user interfaces (UIs), system scalability, and network interconnectivity—than ever before. What more can be piled on top of the never-ending demands and unrelenting complexity of software today? Security of course! It has been demonstrated more than a handful of times that distracted and lackadaisical software developers contribute to a large portion of the security-related flaws present in our software today. Having said that, it can be difficult for even the best developers to avoid security-related mistakes—especially because it's virtually impossible to detect many security flaws without a full system context and analysis.

Unfortunately, attackers, malware, and disgruntled employees exploiting previously untouched software flaws are an increasing threat to businesses and industrialized economies. As applications become more extensible and networks more interconnected, software security vulnerabilities are growing in number. They're also becoming easier to find and simpler to exploit while enabling attackers to easily cover their tracks.

No business wishing to survive in this information age can afford to fall victim to poorly written software. Because of this, no developer looking for a successful career in software development can afford to overlook critical secure programming practices. Trouble tends to trickle downhill and developers will ultimately suffer the consequences as more people learn to see through the marketing and public relations twists often used to cover up current software development problems.

In a perfect world, software developers would stay a step or two ahead of the black hats. Realistically, however, most security-savvy developers are doing great if they're even a step or two behind. The short-term goal of security-savvy developers should be to stay on track with the bad guys and then pull ahead in the race long term.

Integrating security into the coding process—as opposed to layering security protective measures on top—is essential to preventing computer and network attacks. Moving forward, the most successful software professionals won't be the ones writing the most lines of code or designing the fanciest routines but rather the ones that have the foresight to see big picture items (such as security) and the business value they have to offer from a broad developer's perspective.

Q 2.2: What key areas should our development team focus on to ensure the most solid and secure applications long term?

A: First and foremost, developers should focus on building in security as early on in the process as possible. Simply adding security patches on top of existing code is not very effective. This focus is much easier said than done, but even small steps taken now can make a big difference long term. Building in security up front requires current knowledge of software security exploits, secure development practices, and the proper testing tools. Development managers—and developers themselves when possible—should obtain periodic training in secure development and familiarize themselves with and utilize high-quality software security testing tools, as they're the key elements for secure development.

Developers should also utilize widely-accepted security standards whenever possible. This includes Transport Layer Security (TLS) for protecting data in transit and AES encryption for securing data at rest. There are also widely-accepted best practices for developing secure applications—such as the Secure Coding: Principles & Practices homepage (<http://www.securecoding.org>) or the OWASP Top 10 Project (<http://www.owasp.org/documentation/top10.html>)—that provide minimum standards for Web application security that developers need to fully understand. Additionally, special attention is owed to buffer overflows because the majority of software security vulnerabilities are based on them.

It doesn't matter how solid your software applications are if the underlying operating system (OS) is not secure. Therefore, the OS platform (Windows, Linux, UNIX, OS/400, and so on) plays a small role in long-term security as well. For example, the versions of the Windows OS based on the NT platform (Windows NT, Windows 2000, Windows XP, and Windows Server 2003) have certain well-known architectural flaws that have proven difficult to overcome. Likewise, certain hardware platforms can offer protection against software security exploits such as the buffer overrun protection built-in to Intel's Itanium 2 architecture. For new software projects, it would behoove the savvy development team to consider these variables. Standards and best-practices organizations such as NIST and The Center for Internet Security are a good starting point for OS security resources.

The programming language of choice plays an ever smaller role but is something that should be considered. For example, C and C++ tend to have more inherent vulnerabilities than, say Java or .NET. Many languages have inherent security vulnerabilities (C and Visual Basic) that can be difficult, if not impossible, to overcome, while others were designed to be security-friendly with their memory protection and sandboxing capabilities (Java and C#).

In less security-friendly languages, developers may end up spending more time trying to find secure ways of writing code to work around built-in problems instead of focusing on the tasks at hand. Having said all of this, the language selected clearly has certain long-lasting impact on secure coding practices, but no language is immune to security problems regardless of any claims otherwise.

Q 2.3: What are some commonly overlooked software security vulnerabilities?

A: There are several areas that contribute to software security problems yet go unnoticed until it's too late.

Revealing Comments

Oftentimes, developers and quality assurance (QA) staff place comments in HTML, JavaScript, and other easily accessible code that reveal too much into about the application, the system its running on, the development team, and even the organization itself. Information revealed may include passwords, bugs in a routine, personal information about developers, and more. These comments are not necessarily malicious in intent, but they can give an attacker a leg up on breaking into the application or associated systems.

Buffer Overflows

Information security professionals and developers alike agree that buffer overflows comprise a large portion of the software security vulnerabilities in existence. Buffer overflows occur when a program copies data to an address space that is too small to hold everything. Buffer overflows are especially risky in the highly popular C and C++ languages with the *gets()*, *sprintf()*, and *strcpy()* functions. Given that there are so many variables involved, no matter how long and hard we try to prove that a buffer overflow is not possible, there is no way to know for sure. Thus, it is important to not even introduce any variables that can contribute to the problem.

Mishandling Passwords and Logins

There are several ways that developers handle passwords in their software to introduce vulnerabilities:

- Storing passwords in RAM or in cookies that are easily discovered
- Encoding (not encrypting) passwords—especially as part of a URL string—that can be unencoded relatively quickly
- No minimum password requirements or insecure requirements that are susceptible to dictionary or brute force attacks
- Not implementing intruder lockout to thwart application login tampering including dictionary and brute-force password attacks or implementing too short of a lockout time (a few seconds to a minute or so as opposed to 5 to 10+ minutes)
- Login-related error messages that reveal too much information about how the application utilizes user IDs and passwords, such as when an application returns single messages such as “Invalid username” or “Password incorrect” that can give an attacker a leg up

Assumption that Encrypting Data in Transit Means Everything Is Secure

Many developers and even the average computer user believe that just because data is encrypted or otherwise protected in transit means that it's secure. For example, the TLS protocol can certainly prevent someone from reading data as it goes across the wire (or air in the case of wireless networks), but it's not the silver bullet for security. In fact, most security breaches occur against data at rest. In this case, data encrypted in transit just means that you can't see what the attacker is doing! Encrypting data in transit serves as another layer of security, but should not be relied upon by itself.

Not Considering the Network or Operating System Layers

Many software security vulnerabilities are exploited by breaching the network infrastructure or underlying operating system (OS) instead of attacking the applications directly. This calls for consideration of the network architecture and OS configuration in areas such as:

- Where the application resides relative to the public Internet, DMZ, and internal network
- How the application communicates with clients
- How the application communicates with any associated databases (directly with the database on the same network segment or across the firewall to the database in a DMZ)
- Whether the application will reside on the same system as the back-end database

In addition, a really secure application doesn't do much good on an easily compromised OS. Therefore, it is critical to change the default settings to harden the underlying OS (or at least recommend it to the software end user) according to best practices such as NIST's Guidance for Securing Microsoft Windows XP Systems for IT Professionals and Systems Administration Guidance for Windows 2000 as well as The Center for Internet Security's Benchmark and Scoring Tools. Likewise for any associated Web servers, databases, and storage servers.

Software that must run as an OS service or daemon can also introduce vulnerabilities if it uses an administrator, root, or other system-level account. If a vulnerability in the software is exploited, the exploitation can lead to malware or an attacker having full administrative rights on the system to do just about anything. Such operation may be required, but such exploit scenarios need to be considered nonetheless. This is yet another critical issue that highlights the need for increased software security.

Q 2.4: How can software security vulnerabilities be categorized so they're easier to understand?

A: Software security vulnerabilities often mean one thing to developers and something quite different to information security professionals. There are various well-known categories of software exploits. Also, keep in mind, there unknown software flaws that won't necessarily cause immediate problems or crash systems but will likely rear their ugly heads eventually. The following is a high-level categorization of known software vulnerabilities.

Denial of Service

Denial of Service (DoS) occurs when software is overloaded and cannot serve legitimate requests:

- Buffer overflows including integer errors and risky functions such as *strcpy*
- Format string mishandling
- Race conditions and other timing problems that cause a system to get stuck in a loop
- Exception mishandling whereby a problem arises that the program isn't prepared to handle

Authentication Weaknesses

Authentication weaknesses introduce problems that affect the access to software and systems leading to unauthorized usage and breaches of sensitive information. This includes:

- Application logins with no dictionary or brute force attack protection
- Shared user accounts
- Weak or no password security requirements
- Revealing information returned via application error, Web page, or URL strings
- Manipulating login parameters set via unsecure cookies
- Session hijacking whereby an attacker guesses subsequent session IDs and is able to tap into a communications session

Input Attacks

Input attacks occur when user or related system input is not validated for length, content type, and so on which can lead to system crashes or provide access to associated databases, drives, and so on. This includes:

- Login prompt manipulation by overflowing the input buffers
- Login form manipulation including manipulation of hidden fields to change data submitted and cross-site scripting that redirects user input
- Automated input attacks such as completing and submitting forms repeatedly or manipulating email confirmations to obtain unauthorized access
- URL manipulation such as SQL command injection and blind SQL command injection (server error pages are disabled but injection and manipulation occur via inference)
- Command injection to control the application and underlying operating systems (OSs)
- Obtaining and/or decoding information used in GET submissions in HTML forms

Directory Traversals

Directory traversals allow an attacker or an automated system such as a WebBot to browse through directories associated with the software and elsewhere to glean system information and sensitive data or exploit insecure sample scripts. This includes:

- Improper file and directory access controls (by both applications and their underlying OSs) exploitable via FTP, telnet, and more that allow direct data access and system manipulation
- Missing or misconfigured *robots.txt* file that allows for Web server directory traversals to map out entire Web sites
- Path injections to access other system paths and files

Improper Storage of Files and Data

Sensitive information is often scattered about directories with minimal or no access controls. This includes:

- Temporary files that should have been removed
- Text files and even databases that should not be stored in clear text or on the same system as the application
- System configuration files including password files
- Source code including default scripts
- Easily guessed file names

The OWASP Top 10 document is a useful resource for more Web application-specific vulnerabilities. More information can be obtained from the Common Vulnerabilities and Exposures dictionary, which outlines product-specific vulnerabilities in detail.

Q 2.5: Are there any common software development practices that stand out as serious risks?

A: There are hundreds, if not thousands, of ways to develop insecure software, but the most obvious practices can be reduced to a few fundamental factors. Whether a development team has 3 or 30 members and a timeline of 3 weeks or 3 years, the following (listed in order of importance) are risks that no team can afford to take when it comes to ensuring software security.

- Failure to establish formal software security objectives and requirements in advance, such as authentication techniques, data storage methods, input validation, rights management, and so forth to set expectations and get everyone on the same page
- Lack of a unified security front by all development and quality assurance (QA) staff members
- Not integrating secure coding practices in the pre-deployment phases
- Failure to implement secure coding practices during the entire software development life cycle (SDLC) from planning to implementation and beyond
- Lack of acknowledgement of the need for and failure to establish contingency plans for security vulnerabilities that cannot or will not be addressed or remediated
- Using live data in testing and QA that can easily and unnecessarily expose sensitive personal information, intellectual property, and so on

Q 2.6: Are there other technologies or layered security measures we can integrate into our software to help prevent various attacks?

A: The cornerstones of information security are confidentiality, integrity, and availability. As critical as software security is, it's still only one component of a truly secure environment that supports and enables these elements of security assurance. The following are components developers should take into consideration (but not rely upon) when planning for security and integrating controls into their software:

- Network-based firewalls and intrusion prevention systems (IPSs)
- Host-based firewalls and IPSs
- Application-specific firewalls and IPSs
- Strong authentication systems such as smart cards, tokens, and RADIUS servers
- Web server and database access controls and intrusion prevention safeguards
- Operating system (OS) file, directory, and share permissions
- OS hardening using best practices from The Center for Internet Security, NIST, and others

Q 2.7: What role does a layered security defense play in software development?

A: The concept of a layered security defense—often referred to as *defense in depth*—is a fundamental requirement for effective information security. Layered security consists of several controls and countermeasures working in conjunction with one another in a layered fashion with the goal being that if one defense fails or is subverted, other defenses are in place to ensure continued protection of the core system (that is, critical application, database, or other sensitive information). Figure 2.1 provides an example of a generic layered network security defense.

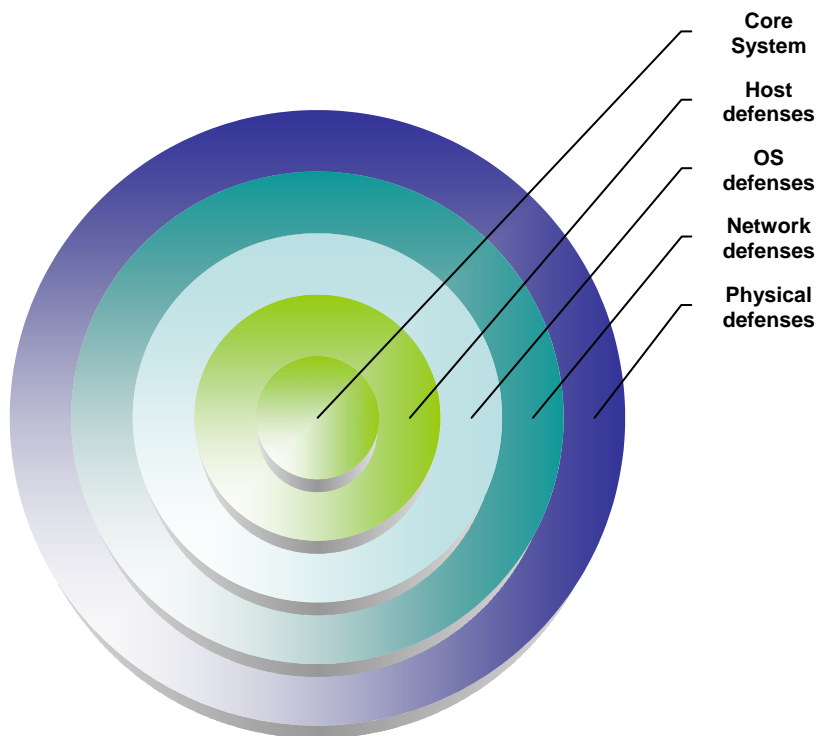


Figure 2.1: Layered network security.

Defenses you might expect to see at the respective layers include:

- Application defenses such as input validation and login timeouts
- Host defenses such as antivirus software and host intrusion prevention systems (IPSs)
- Operating system (OS) defenses such as file permissions and unique user IDs
- Network defenses such as firewalls, IPSs, and virtual private networks (VPNs)
- Physical defenses such as biometrics and security cameras

When it comes to secure software development, a layered defense plays two roles. First, the layered defenses that Figure 2.1 shows can supplement security controls built-in to an application. These defenses are (unfortunately) required to help protect most systems from attack because of the underlying software security flaws. Controls and safeguards such as firewalls and IPSs are often added on after the fact once it's determined that applications aren't secure enough. Too many developers depend on defenses at these levels to compensate for poor coding practices. Doing so is an exercise in futility at best and sets up everyone (developers, users, and network administrators) for failure long term. Regardless of the reasons behind the requirements, such security layers at the host, OS, network, and physical levels are usually required for adequate security.

The second role layered defenses play in secure software development is that the concept of layered protection can, and should be, built-in to software. If any layer of security controls built-in to the user-facing or backend environment fails, software must continue to operate in a secure fashion, and it usually can when layered defense measures are in place. Layered software defenses that developers need to consider include:

- Non-privileged accounts used to run services and daemons
- Secure user authentication
- Secure password management
- Memory wiping where sensitive data was previously stored
- Input validation
- Exception handling
- Error obfuscation
- Secure data storage and transmission
- Any number of security-by-obscurity methods such as running systems on uncommon ports, installing commonly attacked software in non-default directories, and using uncommon configuration filenames

It's absolutely critical for developers not to rely on external security protection but instead create these basic security layers within their software wherever it's reasonable and practical.

Q 2.8: There is a general consensus in my development lab that as long as firewalls and Secure Sockets Layer (SSL) are used, the application is secure—is this true?

A: Many product managers, developers, and even systems administrators have fallen for the *firewall plus encryption of data in transit equals security* myth. This lapse is arguably the most dangerous of all mistakes made when it comes to protecting computers, networks, and the sensitive information they process. Firewalls and data encryption only serve as supporting security measures. A system that solely depends on such protection falls into the “candy security” category, whereby a system has a hard, crunchy outside yet a soft, chewy inside—it’s like having a steel door on a straw hut.

A big problem is that firewalls must be opened for certain applications to be accessible, which often negates most of its benefits. Furthermore, SSL and other means of encrypting data in transit (such as IPsec, PPTP, and WEP) do only that—encrypt data in transit. They provide no protective measures to lock down the software on either end of the communication links. Attackers often prefer SSL and other data communications encryption methods because it masks what they are doing and intrusion detection systems (IDSs) and network-savvy administrators are none the wiser!

Gartner estimates that as many as 70 percent of reported security attacks are at the application level. Today’s networks—especially the Internet and its associated connectivity requirements—don’t necessarily have the best interests of developers in mind. In fact, the scalability of the Internet and large networks seems to be an enabler constantly working against reasonable levels of information security. It’s therefore critical for developers to write software with security in mind—incorporate security layers within their applications where practical—and to perform ongoing security testing throughout the application development life cycle.

Q 2.9: Do I have anything to worry about as long as I develop software with the Open Web Application Security Project (OWASP) Top 10 vulnerabilities in mind?

A: Several of the OWASP Top 10 categories such as Unvalidated Input, Broken Access Control, and Improper Error Handling can be applied to all types of software—client/server applications, embedded systems, and standalone programs. However, the OWASP Top 10 was written specifically for Web application security. Its focus is on common threats and vulnerabilities associated with Web-based systems, which can operate very differently from client/server, standalone, and other legacy applications. Undoubtedly, many newly developed programs are Web-based, but it’s important to look beyond Web application security best practices when developing other types of software.

In addition to the OWASP Top 10, there is another commonly accepted grouping of software security issues found in a book titled *19 Deadly Sins of Software Security* (McGraw-Hill). This piece by software security experts Michael Howard, David LeBlanc, and John Viega goes into detail regarding former Director of The Department of Homeland Security’s National Cyber Security Division Amit Yoran’s theory that “95 percent of all software bugs are caused by the same 19 programming flaws.”

Additionally, there are several excellent books that developers and information security professionals serious about software security should have on their bookshelves:

- *Buffer Overflow Attacks* (Syngress) by James C. Foster
- *Hacking the Code: ASP.NET Web Application Security* (Syngress) by Mark M. Burnett
- *Programmer's Security DeskRef* (Syngress) by James C. Foster and Steven C. Foster
- *Hacking: The Art of Exploitation* (No Starch Press) by Jon Erickson
- *HackNotes Web Security Portable Reference* (McGraw-Hill Osborne) by Mike Shema
- *The Database Hacker's Handbook* (Wiley) by David Litchfield, Chris Anley, John Heasman, and Bill Grindlay
- *Exploiting Software—How to Break Code* (Addison Wesley) by Greg Hoglund and Gary McGraw

Although OWASP Top 10, the 19 Deadly Sins, and the books listed are excellent resources for secure coding practices, it's important to not overlook other lesser-known standards and frameworks in order to improve overall practices throughout the software development life cycle:

- ISO/IEC 12207:1995—Information technology—Software life cycle processes
- ISO/IEC 17799:2005—Information technology—Security techniques—Code of practice for information security management
- NIST Special Publication 800-27—Engineering Principles for Information Technology Security
- NIST Special Publication 800-55—Security Metrics Guide for Information Technology Systems
- NIST Special Publication 800-64—Security Considerations in the Information System Development Life Cycle

Keep in mind that there is no one best resource. Rather than searching for the ideal fit, focus on learning and using a variety of software security best practices, standards, and frameworks and integrate the practices that best fit into your development environment.

Q 2.10: What development practices can we integrate into our daily development routines to reduce the number of security vulnerabilities?

A: Developing secure software is both an art and a science. It's a tricky balancing act for development managers to meet business needs, functional requirements, and time-to-market pressures in addition to eliminating software security flaws on an ongoing basis.

Regardless of the development language used or the underlying operating system (OS), security-focused development practices can and must be implemented throughout the software life cycle. Rather than just learning about secure software development and how to set up a secure test environment—both self-taught lessons or via training classes—and expecting to immediately see quality improvements, it's important to integrate various tools and techniques into the process. This inclusion means performing peer code reviews, running static analysis tools, and carrying out penetration tests on your software throughout the software development life cycle. It may also mean taking advantage of security controls specific to the language and OS used.

Dennis Ritchie and Brian Kernigan, the authors of the classic book *The C Programming Language* (Prentice Hall PTR) stated from the get-go that C is a relatively low-level language. In addition, a large portion of applications are written in C and C++. These languages offer such low-level control and flexibility that they can introduce pretty serious security vulnerabilities if developers aren't careful. Developers must understand that basically all languages (especially C and C++) are, in a sense, working against them. This shortcoming requires developers to become intimately familiar with the nuances of the language(s) with which they're working.

However, a common problem is developers spending too much time on instruction sets and syntax and not focusing on building secure code. Stable code that is well-written according to the language requirements and other best practices does not necessarily translate into secure code. Developers must strike a balance and not overlook the fact that security plays a large part in the overall software quality equation.

Adding to the complexity, anytime developers are forced to learn a new language or drill down further into their current language (for example, making OS, network, or cryptographic calls), the odds of security errors and oversights greatly increase. This situation can be remediated through a combination of experience and automated security analysis tools.

Oftentimes, developers are working on deadlines and have very little time for outside learning—especially for security, a topic that has been out of developers' hands for all too long. Support from management and discipline on the part of developers is needed to make this happen. Even if only a half an hour of ongoing education in software security takes place each day, that adds up to more than 3 weeks of training each year, which can provide great benefits to the developers and their teams. Supporting resources to help developers with such ongoing education include the following:

- *Software Development* magazine (<http://www.sdmagazine.com>)
- Dr. Dobb's Journal (<http://www.drdoobs.com>)
- 2600—The Hacker's Quarterly (<http://www.2600.com>)

These periodicals consistently provide insight into areas such as the latest security testing tools, coding methods, and attacker techniques that anyone wishing to develop secure software needs to read. Developers can also benefit from the security events and Webcasts hosted by Microsoft at <http://www.microsoft.com/events/security/default.msp> and additional resources provided by the Secure Software Forum (<http://www.securesoftwareforum.com/index.html>).

Software security is not very sexy and some people find it boring if not downright annoying. Coding flaws will never go away completely, but major changes can be made through improved development practices over time. Writing secure software is an ongoing and enduring process that requires strong influence from development managers and above. Secure development practices should be seen as an enabler of writing solid applications and improving overall quality long term—something that any developer can be proud of.

Q 2.11: What development tools can we integrate into our daily development routines to reduce the number of security vulnerabilities?

A: Many popular tools for quality assurance (QA) and penetration testing are available but those are often brought into the picture towards the latter part of the development process. It is a best practice to integrate tools into your daily development processes—especially early on when coding first begins. Tools that integrate tightly into development consoles, analyze code line by line, offer security policy customization, include security metrics for measuring quality, and provide detailed recommendations can detect and prevent software defects at the “source” where it counts.

Humans are ultimately to blame for security defects entered into software and many of these defects can be discovered later in the development life cycle. However, there is no realistic way to catch every problem after the fact. The best solution is to prevent security flaws from ever entering the code. In this arena, static code analysis tools shine. These tools are not inexpensive but can easily pay for themselves during the first software project by saving time, money, and resources required to fix security and other software quality problems down the road.

Q 2.12: Is it really worth encrypting our database if we have implemented various security layers in our software? If so, what parts of the database should we encrypt?

A: A layered software security defense consisting of input validation, secure password authentication and password management, and proper error handling can go a long way toward improving software security. However, if it’s determined (via threat modeling or risk analysis) that the database your application uses is susceptible to attack and puts sensitive information at risk, you should consider database encryption.

Database encryption is one of those often-overlooked areas of information security; however, it's becoming a necessity in today's environments and can be an effective protection measure to complement other software security controls. Attacks against sensitive data such as Social Security numbers, credit card numbers, and health information housed in backend databases is a growing business problem. Not only are identities being stolen, credit cards being used in an unauthorized fashion, and private information being leaked (see <http://www.privacyrights.org/ar/ChronDataBreaches.htm> for recent breaches), companies and their executives are facing steep fines and prison time for not taking the proper precautions.

Most of the responsibility for database security is currently in the hands of network administrators and security managers. However, software developers who see the big picture and write their applications with secure data storage in mind can have a great selling point and competitive advantage in the market. Obviously, there are various technical architecture issues related to database encryption, but encryption is an option that developers should consider in the planning, requirements, and design phases of the development life cycle.

Regarding what to encrypt, it depends on how much of a performance hit can be tolerated and how much of the database needs to be protected. Database encryption can cause performance hits that can mostly be alleviated by using a third-party encryption accelerator. It's therefore critical to determine which tables or rows and columns need to be protected and only protect the minimum necessary.

Keep the following practices in mind if you decide to encrypt all or part of your application's database:

- Consider built-in protective measures such as those included in Microsoft's SQL Server 2005
- Determine how you'll design your key escrow
- Avoid storing encryption keys within the application or the database itself
- Clear memory used for key storage once the keys are no longer needed
- Ideally, use different keys for different data sets and for different purposes to minimize losses in the event of a key compromise
- Consider hardware-based cryptographic accelerators or off-loading to a computational server
- Asymmetric (aka public key) encryption has a greater performance impact than symmetric encryption (aka private key)

Topic 3: Auditing for Software Security

Q 3.1: What benefits will a formal software audit offer compared with other types of software security testing?

A: There are various benefits of performing a formal audit over simply testing for security vulnerabilities. An audit in its truest sense compares what is said is being done with what is actually in place. Therefore, a formal audit can help ensure compliance with internal policies and external regulations and verify that secure coding practices are taking place. An audit can also highlight where discrepancies exist in development practices such as one team strictly adhering to security policies and standards while another isn't taking security seriously. Formal audits are also good tools to measure the performance and quality of work delivered by outsourced providers.

Many customers require a software audit (SAS70 or ICA Handbook—Assurance Section 5900) as a part of their due diligence before making a large investment. If you've had a recent software audit, you can use the results for sales and marketing purposes and as a tool for product differentiation. Taking this a step further, a formal source code audit can offer the long-term benefits of industry and governmental regulatory compliance.

Additionally, if such an audit is performed by an independent third-party, the audit can provide information from a fresh set of eyes without bias or political pressures to scrutinize security holes in some areas while skimming over others. A third-party may also have several software auditing tools to look at your source code from different perspectives.

Q 3.2: What are the different types of tests and tools I can use to assess the security of my software?

A: There are four main types of testing methods (two of which are manual and two that are automated) you can use to test software for security vulnerabilities:

- Manual code reviews whereby an individual or team of software security experts sifts through each line of code looking for potential or known problems
- Static analysis tools that analyze source code automatically—often much more quickly and efficiently than a human would be able to. There are both freeware/open source and commercial tools that can be language-specific tools (C or Java-focused) and all-in-one (can review most popular types of source code)
- Manual security assessment whereby an individual or team of information security experts pokes and prods the application in a real-world environment looking for exploits
- Real-time analysis tools that automatically analyze the working software in a real-world environment—often much more quickly and efficiently than a human would be able to. There are both freeware/open-source and commercial tools that can analyze standalone programs, client/server applications, Web applications, and database systems

The different methods and tools have advantages and limitations and can complement each another. There is not a best or comprehensive way to test for all software security vulnerabilities. However, each method plays a critical role in software security.

Q 3.3: How can I determine whether we should perform a software security audit?

A: There are several factors that help determine whether a software security audit should be performed. This decision is strictly up to each business, development team, or individual developer, but some key factors driving such audits are:

- Internal policy requirements
- Industry or governmental (state and/or federal) regulations and laws
- General market demands
- Customer or business partner contractual requirements
- Mergers and acquisitions
- Outsourced development
- Limitations on security expertise and resources

If formal security testing such as penetration testing or static analysis has never occurred and your organization cannot afford to sell or use software that is vulnerable to myriad security problems, a software security audit is highly recommended.

Q 3.4: Who should perform formal software security audits?

A: Who performs a formal software security audit for your organization depends on your business needs and what you want to get out of the security audit. The following people serve equally important roles in auditing the security of your software:

- Internal auditors seeking to assess security controls for regulatory or internal policy enforcement purposes
- Business partners looking to determine the viability of your software and the value it can add to their business endeavors
- Customers looking to determine how your software can meet their needs and how secure and compliant with government regulations it will help them become
- Independent external auditors you hire to audit the controls built into of your software based on internationally accepted best practices such as COBIT
- Independent information security consultants you hire to assess the security of your software from an outsider's perspective based on internationally accepted best practices such as the newly revised ISO/IEC 17799:2005 and the OWASP Top 10

The person that should ultimately perform your software audits is the person who has a contractual or legal obligation to do so or is the person with the right skills you need to assess the security of your software.

Q 3.5: Which areas of our software should we audit?

A: Ideally, every aspect of your software should be audited for security vulnerabilities, but this ability is not always realistic or necessary. You or your auditor may have a pre-determined methodology for auditing software. At a minimum, software audits should cover security best practices and standards as much as possible.

The critical areas to concentrate on are as follows:

- Source code
- Authentication methods
- Password management
- Input validation
- Exception handling
- Data storage methods and safeguards used
- Data transmission methods and protocols and safeguards used
- Proper packaging in order to hide configuration details—especially with regard to Web applications

You might want to test your software's resiliency to Denial of Service (DoS) attacks. If so, proceed with caution as many DoS tests can cause system hang-ups, crashes, and so on that can lead to data corruption unwanted downtime.

Keep in mind that a program's overall logical structure isn't necessarily defined by its source code alone. Additional layers, services, computers, and networks are often in the mix complicating matters. Therefore, secure standalone code doesn't always translate to a secure application or system. Also, odds are that security vulnerabilities are present in various runtime modules of your software—not just in the code that handles your software's security controls.

At a higher level, an area often overlooked during audits is the actual development process. If time, money, and resources permit, this area is useful to review to uncover vulnerabilities that might have otherwise been overlooked.

Q 3.6: What are some common software security auditing mistakes?

A: There are many software security auditing oversights and mistakes that can have a serious impact on the quality of software. First, looking at each piece or unit independently without considering the big picture is a big mistake. In other words, if you don't review how the code interacts and operates with the hundreds and often thousands of variables present in today's networked environments, many exploits can go unnoticed. Thus, it's critical to utilize more than one auditing method and look at your software and systems from more than one perspective.

Another common mistake is relying too much on automated tools to find everything. There needs to be human involvement testing the software manually. Keep in mind though, that manual code reviews or penetration tests aren't the complete solution either. With today's complex software and network environments, there are simply too many variables and too many tests for any human being to possibly handle.

Many people often overlook the value of code reviews. Likewise with penetration testing—they complement each other and play a significant role in finding the largest number of vulnerabilities possible.

In addition, many software security auditors rely too heavily on checklists. Best practices and software security frameworks form a good basis for testing, but the most vulnerabilities will be unveiled by experienced (and most often) technical auditors that understand networks, computers, and software at the packet- and bit-levels.

Finally, perhaps the biggest mistake of all is when software security auditors believe that all possible testing scenarios have been executed and all security flaws have been discovered. Again, technical expertise and system complexity play a major role here. Thoughts of 100 percent software security are delusional at best. This problem is mitigated by using static analysis tools.

Q 3.7: How can I determine whether we need a formal software security audit or other type of test such as a manual code review or penetration test?

A: If software security and software quality are a priority, all three types of reviews need to be performed. However, the type of testing you can perform depends on the type of software you have. For example, you won't have the source code to test for an application that is a commercial off-the-shelf product. This situation leaves certain aspects of security up in the air because you cannot test from every perspective. Of course, if your license agreement doesn't forbid it, you can always reverse engineer an application using a disassembler such as IDA Pro (<http://www.datarescue.com/idabase/index.htm>). However, custom software makes it easier to look for coding problems, but you won't be able to assess how the software runs and which vulnerabilities are present in every possible environment.

The major downsides to each type of testing are:

- Formal software security audits are often limited in scope and technical details
- Penetration tests only uncover security flaws during runtime and often cannot get to the heart of the problem—the source code
- Automated code reviews only uncover security flaws at the source code level and may not point to issues that can be exploited when the software is running in a real-world setting
- Manual code reviews can be extremely difficult to perform—especially on large, multi-function applications—and flaws are easy to overlook

Keep in mind that although each method has its pros and cons, all three are necessary to ensure the best assessment from all possible angles.

Using a real-world example, buffer overflows and other memory errors are an elusive and widespread problem. In fact, it's a well-known fact that most security vulnerabilities are based on memory allocation problems. Further complicating the issue is when one security flaw such as a buffer overflow is combined with reentrant code to lead to something exponentially more dangerous. This problem in and of itself calls for looking at software security vulnerabilities from as many perspectives as possible including static analysis, run-time analysis, and even compiler options and other memory error detection tools when possible.

Q 3.8: What features should I look for in software security testing tools?

A: The criteria for shopping for software security testing tools depends on which type of testing tool you'll be using. The following list provides generic features and characteristics to look for at a minimum:

- Platform and language support
- Ease of use
- Reporting
- Automatic updates for new vulnerabilities and program patches
- Customizable security policies
- Ability to prioritize software vulnerabilities discovered
- Well-known and respected in the information security and software development industries and is willing to provide customer references
- Long-term viability of the vendor
- Acquisition, operational, and ongoing support costs

The easier a tool is to integrate into the auditing, testing, or development process, the higher the likelihood that it will be utilized. Tools that are inconvenient or offer little value are likely to be overlooked or frowned upon in the name of software development and security testing efficiency.

Q 3.9: Would a software security audit performed by an external and independent consultant produce better results than one performed by our own internal IT auditing staff?

A: Not necessarily but it can be beneficial. Table 3.1 highlights the pros and cons of outside expertise to perform your software security audits.

Pros	Cons
Provides a fresh look at your source code and application	An outsider will not know your software like your team does
Less chance of bias due to political influences	The necessary knowledge transfer may not occur due to resentment or politics
Can save time and resources to let your team focus on other tasks and areas of expertise	May require a considerable upfront investment
Outside auditors can often spread the cost of their security assessment tools across all their clients, which helps minimize the costs of that portion of the testing project	It can be difficult to find professionals with extensive knowledge in both development and security

Table 3.1: The pros and cons of using an outside expert for software security audits.

This decision is ultimately a business decision that depends on varying factors. If the decision is made to hire an outsider, be sure to consider the following:

- Cost compared with performing the review in-house
- Experience of the auditor
- Technical knowledge of the auditor
- Tools the auditor will use
- Security and software development certifications
- Customer references
- Sample deliverables

Q 3.10: Are there specific software security standards and best practices we can look for to ensure we're getting the most from our auditing investment?

A: Although not absolutely necessary, it can be beneficial to have your software security auditor perform an audit against various standards and frameworks such as the following:

- OWASP Top 10 (<http://www.owasp.org/documentation/topten.html>)
- ISO/IEC 12207:1995—Information technology—Software life cycle processes (<http://www.iso.org>)
- ISO/IEC 17799:2005—Information technology—Security techniques—Code of practice for information security management (<http://www.iso.org>)
- NIST Special Publication 800-27—Engineering Principles for Information Technology Security (<http://csrc.nist.gov/publications/nistpubs/800-27/sp800-27.pdf>)
- NIST Special Publication 800-55—Security Metrics Guide for Information Technology Systems (<http://csrc.nist.gov/publications/nistpubs/800-55/sp800-55.pdf>)
- NIST Special Publication 800-64—Security Considerations in the Information System Development Life Cycle (<http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf>)

Also look for certifications that can be beneficial to you—especially if you have to share your auditing reports with upper management, business partners, or clients. Just because a security auditor is certified doesn't mean they're necessarily qualified to do the job.

Basic security and audit certifications to look for include CISSP, SANS GIAC, Certified Ethical Hacker, and CISA. Software development certifications can be a bonus—especially when combined with security certifications, so look for certifications such as Microsoft's MCSD, MCP .Net, and MCDBA as well as Sun's SCJP, SCJD, and SCEA.

Q 3.11: Is a source code audit all that is needed to identify the big picture software weaknesses as well as granular vulnerabilities?

A: A source code audit is an excellent way to find both high-level design flaws and deeply ingrained software security vulnerabilities. In fact, it's arguably the best way overall to root out security flaws that might not otherwise be discovered through other types of security audits until it's too late. Such is especially true when automated source code auditing tools customizable with various security policies are used.

The downsides to an audit may involve the auditor not being technical enough to delve in deeply or the scope being limited to a high-level set of best practices to be compared against. A comprehensive software security audit would include other types of testing—particularly manual assessments and runtime assessments such as penetration tests.

Q 3.12: What should we do once security vulnerabilities have been identified by an audit?

A: Not all vulnerabilities and related flaws are on the scale, so the first step is to prioritize your vulnerabilities. Four key factors play into this prioritization:

- Impact if the vulnerability is exploited
- Likelihood of the vulnerability being exploited
- Cost to remediate
- Time to remediate

This analysis may require the expertise of someone who understands not only the vulnerabilities and underlying technical complexities but also the basics of risk management. This analysis will likely require input from several individuals, including those outside of development working in information security or a similar functional area.

Once vulnerabilities have been identified, the typical software change control and configuration management steps should be implemented, and the problem should be addressed. The IT Infrastructure Library (ITIL) documents used for implementing an IT Service Management (ITSM) framework (see <http://www.itil.co.uk> and <http://www.itil-itsm-world.com> for more information) would be beneficial here.

All of these security audit follow-up tasks take time, money, and staff resources to complete. Therefore, it's critical to involve management (development management, security management, and other management as needed) to ensure they're completed in a timely fashion and on a reasonable budget.

Topic 4: Testing for Software Security

Q 4.1: What methods are available for my developers and quality assurance staff to do their own software security testing and what are the differences between each method?

A: There are two main options for developers and quality assurance (QA) staff:

- Performing manual code reviews
- Performing automated static analysis testing

The first method is a good way of finding security problems you may not have otherwise discovered; however, it's very time consuming and tedious work and not a highly effective method for achieving secure software. Such is especially true if you have limited development and QA resources.

The second method is much more time-efficient and cost-effective way for developers and QA staff to check and cross-check their own work. If you're spending time to review code throughout the software development life cycle (SDLC)—especially during pre-deployment—it's only logical to use such a tool. Security testing efficiency can be enhanced even more by using static analysis tools that can be integrated into integrated development environments.

Q 4.2: Who should perform software security testing?

A: Everyone from system architects to developers to quality assurance (QA) personnel needs to be involved in the software security process throughout the software development life cycle (SDLC). This includes:

- Planning
- Specifying requirements
- Analysis
- Designing
- Developing
- Testing
- Implementing
- Maintaining

If testing is performed only by QA personnel towards the end of the life cycle during testing or validation, it's too late in the game to effectively and efficiently fix security problems. Developers and QA staff members in conjunction with security experts and even higher-level security auditors need to be included in the process starting with the planning phase. The longer you wait to find security holes, the more difficult it is and the more costly it will become to fill them.

Q 4.3: What software security testing techniques should my quality assurance engineer be utilizing?

A: Security testing techniques are often very detailed and dependent on the software's architecture and complexity as well as the level of expertise of the tester. Certain generic quality assurance (QA) techniques can detect security problems, but there are four main areas the QA engineer can focus on to ensure the critical areas are covered:

- Consider every layer of the software when possible—test all features, not just the main features
- Test for all success and failure paths and buffer limits where possible—attempt to create both expected and unexpected failures; automated tools can help find the latter
- Look for patterns in the code's logic that highlight potential vulnerabilities, such as how certain authentication methods lead to validation of input or how memory is allocated based on user type or program timing
- Cross-check to verify each other's work if resources permit

Keep in mind that there is no way to reasonably expect that all security flaws will be uncovered—especially during the QA testing phase. Chances are that certain security vulnerabilities will be overlooked and may or may not rear their ugly heads down the road. Once secure coding practices are built-in from the ground up, this won't be as big of an issue.

Q 4.4: What software security tests should be performed during a penetration test?

A: Penetration testing is about poking and prodding to glean information that can be used as a stepping stone to glean more information and so on until the opportunity arises that the system can be 'broken into.' Just like black-hat hacking, there are no steadfast rules that apply other than the basic methodology of:

- Gathering general information about the application and the system its running on that can be used a starting point
- Mapping out the software and system to obtain a general idea of the layout and functionality
- Scanning the system to see which ports are open, what login banners may be present, and how it generally responds to requests
- Testing for alternate paths and filenames—not just the defaults
- Looking for specific vulnerabilities
- Exploiting the vulnerabilities found in order to gain access to the system or crash it altogether

During these phases, the penetration tester would assess various software components such as authentication methods in use, password requirements, whether input validation is used, exception handling, and how data is transmitted. These quick tests will likely reveal a lot of information about the software that can, in turn, be used against it for further exploits.

Q 4.5: What are the benefits of performing static analysis code reviews compared with runtime penetration testing?

A: Both static code analysis and runtime penetration testing are essential to obtain an in-depth view of software security flaws. Penetration testing is a great method to test for vulnerabilities from an attacker's perspective. This method offers a tangible way of measuring the security of your software in a real-world environment. However, static analysis tools may not assess the software architecture, and without such an assessment, the scope and results can be limited so ensure the product you're considering can do this.

By analyzing the source code of software, you're able to look deep at the heart of the system. You can see how memory is utilized, how data is processed, and most importantly, you can assess security vulnerabilities in their most basic form.

Different software flaws will likely be discovered during each method. It's the combination of the two methods that can reveal information that enables you to go back and fix software security problems at the source—that is, from the very beginning of the software development life cycle (SDLC). This information will also provide feedback and insight into the development process itself, which can be used to make improvements over time.

Q 4.6: What steps should be taken to ensure the security vulnerabilities found during the testing phase are properly addressed?

A: Consistent follow-up and change control is necessary to effectively plug the software holes that are found during testing. First, you need to organize and prioritize your findings into high (likelihood of exploit and impact are great), medium (likelihood of exploit and impact are considerable), and low (likelihood of exploit and impact are minimal). This is a type of mini information risk assessment.

Once the security vulnerabilities are assigned a priority, formal change control steps (preferably via a formal change control committee) must be taken to determine how the changes will be implemented (as with any other software changes) perhaps similar to the following:

- **Approved**—Will adopt the change immediately (many of the high-priority vulnerabilities fall into this category)
- **Deferred**—The change is valid but may adversely affect the project and will be tabled for now (many of the low and some of the medium priority vulnerabilities fall into this category)
- **Field fixed**—The change may be valid but not critical enough for a formal change in the general code and will be fixed on an ad-hoc basis
- **Refused**—The change does not appear to be valid and provide business value

The next step is to actually fix the flaws you find—especially the high- and medium-priority ones—based on whatever timetable works for you and/or your customer. Consider retesting if you must to clarify any findings. Also, document any lessons learned for future reference. And finally, test again during all development stages where possible and on a periodic basis after general release.

Q 4.7: What is the difference between traditional testing and testing for security?

A: Traditional software testing involves looking at practically every component for functionality, usability, integration, and more—all but the most basic security testing is not considered. The formal high-level categories for traditional testing are:

- Unit
- Functional
- Entrance
- Alpha (that is, system)
- Beta
- User/customer acceptance

When testing for security, the development and quality assurance (QA) team will perform the listed steps and tests as well as integrate certain security tests (static analysis and manual reviews) into the process. The earlier security testing begins, the earlier flaws can be found. Code changes and additions combined often introduce new security vulnerabilities into the process. This highlights the importance of using testing tools that are easily accessible to—if not integrated into—the development environment.

Manual and automated real-time analysis such as penetration testing may come later in the process such as the alpha or beta stages and beyond. This can provide a more real-world view of the software's operation; however, given the time-to-market deadlines of most software, certain security vulnerabilities may be found too late in the game to remediate before general availability. Thus, it is crucial to employ solid development practices from the very beginning.

Q 4.8: Are there any testing tool features that should exist above and beyond those needed to perform higher-level audits?

A: There are literally hundreds of beneficial features and selling points to be considered before making the investment in your software security testing tools. The following features provide the most flexibility and testing capabilities for developers, security auditors, and other software testing personnel:


- Integration into the development process and environment (how certain static analysis tools can integrate with Visual Studio .NET)
- Ability to test various components and multiple layers of security (the in-depth testing capabilities of Web application tools used for penetration testing)
- Graphical visualization of an application's structure
- “What if” modeling for system architecture purposes
- Prepackaged security vulnerabilities
- Automatic updates for new vulnerabilities and program patches

- Ability to determine baseline defects and monitor for subsequent defects introduced through regression
- Ability to keep an audit trail of software defects and associated fixes
- Ability to examine an application's CPU cycles, network utilization, and disk space requirements in order to detect and ward off potential Denial of Service (DoS) attacks.
- Customizable security policies
- Ability to exchange data and integrate with complementary testing tools
- Use of software security metrics to measure policy compliance
- Ability to prioritize software vulnerabilities discovered
- In-depth reporting for both technical personnel and high-level management

Such features cost money and often go unused; therefore, look for features that you and your team members can best utilize for your specific needs.

Q 4.9: Is it better to have an external and independent expert perform software security testing?

A: As mentioned in Q 3.3, the answer depends on various business needs such as expertise required, availability of internal resources, desired tools, and more. You may not have a choice in the matter, especially if a business partner, auditor, or client wants to bring in an independent and unbiased set of eyes.

 See Q 3.3 for information about the pros and cons of hiring an independent auditor.

One of the biggest problems with testing for software flaws internally is the threat of a conflict of interest. The developers who write the code may not necessarily do what is in the best interest of the organization if it means that finding numerous flaws makes them look bad. By hiring an independent outsider, the risk of this situation is much lower, and someone who has not been caught up in the project can often see things that might otherwise be overlooked. Such is especially helpful for finding vulnerabilities and poor programming practices that aren't necessarily exploitable at the moment but can pose problems down the road. Regardless of who does the work, it's critical to have someone who understands the importance of software security, has the proper experience and tools, knows how to interpret the findings of the tools they use, and can make practical recommendations that make good business sense.

Q 4.10: What software security testing tools are the best fit for quality assurance (QA) professionals?

A: Automated static analysis tools are extremely efficient at reviewing millions of lines of code in a very short amount of time and are likely the best pick for QA staff members. They're not perfect, but such tools are the only effective way to review complex software for security vulnerabilities and detect them at the source.

QA professionals need to test both built-in security functions as well as how the software handles exceptions and unexpected input. Given the nearly infinite input variables, it's imperative to have other types automated security testing tools to assist. This arena is where the myriad open source and commercial hacking tools come in handy. The following list highlights tools that are widely used and well-respected among penetration testers and other information security professionals that QA staff could certainly benefit from:

Freeware Tools

- Paros Proxy (<http://www.parosproxy.org>) for trapping and modifying Web requests to check for application vulnerabilities
- Wikto (<http://www.sensepost.com/research/wikto>) for assessing Web applications and server misconfigurations
- Nessus (<http://www.nessus.org>) for assessing network, operating system (OS), and other software vulnerabilities
- Brutus (<http://www.hoobie.net/brutus>) for testing software authentication systems and cracking passwords
- Metasploit (<http://www.metasploit.com>) for exploiting vulnerabilities found in OSs, applications, and databases

Commercial Tools

- QualysGuard (<http://www.qualys.com>) for assessing network, OS, and other software vulnerabilities
- WebInspect (<http://www.spidynamics.com>) for assessing Web application vulnerabilities
- AppDetective (<http://www.appsecinc.com>) for assessing database vulnerabilities

Q 4.11: How are penetration testing tools different from typical software security testing tools?

A: Penetration testing tools are quite different from typical static and runtime analysis tools. Penetration testing uses an ‘anything goes’ methodology, so the various types of penetration testing tools are virtually unlimited. The various high-level categories of penetration test tools is as follows:

- Web search engines such as Google (<http://www.google.com>)
- Internet registries such as ARIN (<http://www.arin.net>) and Whois.Net (<http://www.whois.net>)
- Corporate Web sites that divulge company, network, and application information, including user manuals, knowledge bases, and patches
- Port scanners such as SuperScan (<http://www.foundstone.com/resources/proddesc/superscan.htm>) and Nmap (<http://www.insecure.org/nmap>)
- Application mapping tools such as Amap (<http://thc.org/thc-amap>) and SMTPscan (<http://www.greyhats.org/?smtpscan>)
- Vulnerability assessment tools such as QualysGuard (<http://www.qualys.com>), STAT Scanner (<http://www.stat.harris.com>), and Nessus (<http://www.nessus.org>)
- Password crackers such as Proactive Password Auditor (<http://www.elcomsoft.com/ppa.html>), Cain & Abel (<http://www.oxid.it/cain.html>), and Brutus (<http://www.hoobie.net/brutus>)
- Network analyzers such as EtherPeek (<http://www.wildpackets.com>), CommView (<http://www.tamosoft.com>), and AirMagnet Laptop Analyzer (<http://www.airmagnet.com/products/laptop.htm>)
- Network protocol hacking tools such as ettercap (<http://ettercap.sourceforge.net>) and dsniff (<http://naughty.monkey.org/~dugsong/dsniff>)
- Web application vulnerability assessment tools such as WebInspect (<http://www.spidynamics.com/products/webinspect/index.html>), Hailstorm (http://www.cenzic.com/products_services/cenzic_hailstorm.php), and Wikto (<http://www.sensepost.com/research/wikto>)
- Web application proxies such as Paros Proxy (<http://www.parosproxy.org>)

The problem with most of these tools is that they all run as separate programs and don't communicate with one another, which can make the software vulnerability detection and correction process more difficult to manage—especially if multiple QA engineers are working on the same project. However, these tools are still a vital component of an effective software security testing program.

Complementing penetration testing tools, software analysis tools typically look at source code or specific problems within an application's executables and runtime libraries. Some tools are standalone applications and others snap into integrated development environments. There is arguably certain overlap when it comes to Web application tools, but by and large, there is a significant difference between penetration testing tools and traditional software testing tools. Each have their own place and each are suited for different types of security professionals.

Q 4.12: Should we focus time, money, and efforts on performing code reviews, penetration tests, or both?

A: “Testing” as we've known it to this point has mostly been ignored when it comes to application security, but that paradigm is changing. People on the bleeding edge of software security are realizing that there is no one best solution for assessing software vulnerabilities.

Manual code reviews can be beneficial to help ensure secure software, but they cannot be relied upon solely. Most developers and security experts are not qualified to look for security flaws embedded in complex code—especially where there are thousands and often millions of lines of it to pore over. Performing manual code reviews on anything outside of a couple of thousand lines of code is too arduous to be effective.

Static analysis tools are extremely efficient at finding flaws very quickly that would otherwise take unrealistic hours to find or would be overlooked altogether. Although a vital component of secure software testing, they cannot be relied upon solely because they can't see the application running in real-world settings.

Penetration testing is an excellent way to test an application in a real-world environment and there are plenty of tools to facilitate the process. However, the penetration tester may overlook certain areas of an application or stop once one or two exploitable vulnerabilities are discovered. It's difficult to find deeply embedded security flaws through penetration testing, which can lead to a false sense of security.

Q 4.13: What are some common oversights and mistakes made once the software testing phase is complete?

A: Two major items are often ignored for varying business, technical, and political reasons. For starters, a concept called *threat modeling*—which should actually take place before the testing phase—is often overlooked. Threat modeling is the process of identifying sensitive information that the software accesses, processes, or manages and then determining the threats that can exploit such information. Developers, information security experts, and quality assurance (QA) personnel need to think like an attacker analyzing the software looking for obvious and not-so-obvious attack points. Once the weaknesses are found, countermeasures can be developed and put in place where practical. If threats and vulnerabilities are serious, it may warrant a serious software overhaul or re-design.

Another common problem that can throw a wrench in the software security assurance process is failure to address the security flaws once they're identified. Identifying security flaws is one thing, but actually doing something about them is quite another. It's easy for developers and development managers to become busy putting out other fires and push security assessment findings to the side. The only effective way to ensure the issues are properly addressed is for management to lead the process—which includes software security being on the radar of upper management. In addition, it's important to utilize software quality metrics to ensure that security is being dealt with properly and that the overall development life cycle is treated like any other important business function.