Encyclopedia of Information Systems and Technology

Phillip A. Laplante, Vladimir Stantchev, Jeff Voas, Anne-Marie Smith, Qiuming Zhu, Morris Chang, Irena Bojanova, David Alan Grier, Jia Zhang, Thomas Costello, San Murugesan, Jim Goldman, Joanna DeFranco

Software Development Lifecycles: Security Assessments

Publication details
George G. McBride
Published online on: 29 Dec 2015

Accessed on: 20 Aug 2023

PLEASE SCROLL DOWN FOR DOCUMENT

Full terms and conditions of use: https://www.routledgehandbooks.com/legal-notices/terms

This Document PDF may be used for research, teaching and private study purposes. Any substantial or systematic reproductions, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.
Software Development Lifecycles: Security Assessments

George G. McBride
Senior Manager, Security and Privacy Services (SPS), Deloitte & Touche LLP, Princeton, New Jersey, U.S.A.

Abstract
With events, such as buffer overflows, Structured Query Language code injection, and arbitrary code injection, we are faced with a continuous flood of vulnerability and threat information for our systems, our applications, and our networks. Whether the information comes from a customer, an employee, or an auditing or assessment firm, organizations are continuously addressing the endless cycle of vulnerability and threat identification, measurement of risk, and the implementation of some appropriate corrective action (also referred to as a control). Surely, there must be some measures that organizations can take when developing software to proactively address security and in turn reduce potentially negative publicity and the costs of development and ongoing maintenance for themselves and their customers.

INTRODUCTION
This entry discusses how organizations that are involved with the development of software systems can build security, reliability, and resiliency into their applications. In addition, readers of this entry will also understand areas that should be reviewed during an audit or assessment of a typical software development life cycle (SDLC). The software engineering field has several equally viable and applicable SDLC methodologies depending upon the business, industry, type of application, and experience of the development team. This entry provides recommendations, best practices, and areas to review during an audit or an assessment for any and all of the SDLC methodologies. Finally, every effort has been made to ensure that whether you develop in house or outsource the development of software systems, each aspect of this entry will be relevant to you.

This entry focuses on the following areas:

- The need for secure and reliable software
- Development environments, including physical and logical security, source code management, auditing, authentication, authorization, and access control to source and run-time code
- Common security challenges to all SDLC methodologies
- Security with purchased, open-sourced, and proprietary code embedded in applications under review
- Security in the requirements and definition phases
- Security in the software systems design phase and how Formal Methods can help secure the design
- Security in the implementation and coding phases, including source code review tools
- Security in the integration and testing phases including module and unit testing and integration
- Security during installation and deployment phases
- Security in the lifecycle maintenance mode, including software updates, obsolescence, and decommissioning
- Security through third-party solutions, and whether they hinder or help the overall software solution

One of the first questions that any fiscally minded manager may ask is “Why?” Why would any company choose to spend additional funds, accept longer development cycles, and possibly require additional personnel to develop code more securely when customers are already buying the software as is? Perhaps the thought is that the initial code will be developed and shipped, and then security features will be implemented as incremental updates over the product’s lifecycle, thus ensuring a first-to-market strategy. Perhaps the thought is that nobody will notice the absence of security features, or that the security features will not be required as the software is not mission critical, or will not be associated with any sensitive data.

Whether the use of the software exceeds its programmers’ original expectations, whether it is run on a platform on which it was not originally intended to run, or whether the system receives input data from systems and processes that were designed years later, there is little in today’s system design that developers can trust or assume.

Finally, one of the strongest reasons for building security into today’s products during the development cycle (and not post-deployment) is cost savings or cost avoidance, depending on your view. For the consumers, whether it is an individual or a business, there are costs associated with applying patches, hot fixes, updates, or
service packs. Connection charges to receive the patches, time taken away from other activities, business disruption, building install packages for the patch, regression testing, and increased network bandwidth are just some of the additional “costs” to the purchaser. The companies that produce software with security defects have costs as well. In addition to making sure that their own infrastructure maintains the latest patches and updates for their applications and operating systems, they also incur costs associated with the management of the software vulnerabilities in their own applications. Longer maintenance cycles, additional personnel, additional testing, additional patches, and the erosion of the company’s base or brand name are all additional costs borne by the manufacturer.

Performing an internal code walkthrough during the design phase, discovering a vulnerability, making a few changes to a few lines of code and updating the documentation (if that is even necessary) could take as little as a few minutes. Having the help desk field calls from concerned customers who believe that there is a security vulnerability, logging the issue into a database, having a quality assurance associate duplicate the problem, opening up the code, reviewing the code, updating the code, updating the documentation, packaging the update, maintaining the new version, shipping it out, and then fielding calls from customers wondering why the patch just disabled some other application, will cost a lot more. In today’s environment, it is not a matter of if the costs will be incurred; it is a matter of when and how much. Nobody can argue money can be saved by fixing an undocumented feature (a software bug) or vulnerability after the first vulnerability is detected and the product is already in the hands of the customers.

Likewise, there are several reasons why security features (and other features, such as privacy, reliability, resiliency to disasters, etc. that will be discussed later) are not typically incorporated in the systems that are still being developed today. Lack of awareness continues to be the reason most given as to why vulnerabilities continue to exist in code. Even with all of the advertisements, supporting applications, magazines, books, and announcements seen today, software developers often feel that they are not at risk for a number of reasons, such as assumed external controls, assumed validated input and so on. Security features, similar to any other feature or requirement, cost money to implement, time to design, code, and test, and may be considered too restrictive to the application from an end-user experience.

Why not just build security features into applications today? Why not just run some tools and ensure that every software bug, whether security related or not, is mitigated? Software design is an inherently complex process, with multiple programming languages, development methodologies, and development environments. Owing to continually evolving development and compiler aids, oftentimes there are an infinite combination of inputs and platforms to run on, which further amplifies design complexity. However, it is not an impossible task, and the remainder of this entry highlights the activities that a development organization can undertake to increase the security and reliability of its applications.

SOFTWARE DEVELOPMENT LIFE CYCLE

There are a number of SDLC models in use today. Waterfall, spiral, rapid application design, joint application design, and prototyping are five of the more common models used by programmers and software engineers when developing software projects. The model chosen is typically dependent on the size of the project (either the team or the size of the expected code base), the amount of time available, how firm the requirements are, and the background, familiarity, and experience of the design company and its employees. Although any model is capable of producing secure code, without strong controls, some models may be more disposed to producing less secure code. For example, the waterfall model maintains strong gates between each of the development cycles, whereas the rapid prototyping methodology usually involves several iterations between end-users (or the marketing organization) and the development team to reach an agreement on the look, feel, and high-level functionality of the application. Once an agreement has been reached and the requirements have been defined, the prototype is supposed to be discarded and the development efforts are begun from scratch, based on the requirements developed during the prototyping activities. How many organizations do you believe actually do that?

SOFTWARE DEVELOPMENT SECURITY FUNDAMENTALS

The guiding principles of the software development process should be documented in a hierarchically arranged and integrated set of policies, practices, standards, and procedures. This policy framework should document many aspects of the SDLC, such as the following:

- A policy that states that the prototyping development methodology will be utilized in all customized software development efforts
- A practice that defines how a particular code is commented
- A standard that identifies the permitted programming languages or development environments
A procedure that provides step-by-step instructions on how to conduct a code review or generate a software build.

It has been my experience through many audits and assessments that the policy framework might exist, but may be antiquated and not used because it adds no value to the overall process. An up-to-date, well-maintained policy framework provides the foundation and guiding principles for defining how software is developed securely, efficiently, and within company standards. In the event of a disaster, an up-to-date policy framework could be utilized to support recovery operations. Additionally, a policy framework is required to support auditing activities, ISO certification, and other compliance-related activities. The need for a SDLC policy framework is inevitable. Why not ensure that your framework is up to date and complete now, and use it to drive development activities, rather than completing it after the fact to prepare for an audit?

The waterfall model is one of the most documented and most structured development methodologies available and will be used as an example throughout this entry. There are several phases of the waterfall model, including:

- Business case and conceptual requirements definition
- Functional requirements and specifications definition (what it needs to do from a business perspective)
- Technical requirements and specifications definition (what it needs to do from a technical perspective)
- Design and architecture of the system
- Coding
- Unit and system test
- Implementation and deployment
- Maintenance
- Decommissioning of software systems

A typical software design team has several coders, one or more architects or software engineers, some quality assurance personnel, a team leader, a project manager, user representatives (sometimes marketing personnel), and sometimes a secretary or recorder who is responsible for taking notes and minutes. Typically missing from most teams is a security consultant or advocate who can offer guidance, support, and advice on security issues throughout the SDLC. In the absence of that advocate, this entry provides introductory advice the development team can use to add some baseline security functionality to the next release.

Securing the Foundation

One of the most commonly overlooked areas is physical security, and it is important to cover a few basic concepts in this entry. At a very high level, we should be concerned about the physical security of the developer’s workstations, as well as the security of the source code repositories, build machines, source code back-up, and so on. As any lawyer will tell you, the more that you protect your intellectual property, the easier it will be in court to prosecute somebody who has inappropriately gained access to it. If you leave your code stored on several developers’ machines, burned on CDs lying around, and printouts of code in the development labs, opposing counsel will always ask “How valuable could it have been?”

If you can perform a thorough physical review, conduct one from top to bottom. If you cannot, at a minimum, the following should be done:

- Ensure that back-up tapes of source code, sample data, and design documents are conducted regularly and properly secured
- Take the clean desktop policy to heart and ensure that all electronic media and paper copies are properly secured at each developer workstation
- Review the physical security of the server room (and perhaps of the developers if they are co-located in a single area) to include access controls, logging, environmental controls, guest access, etc.

Likewise, a team of information technology (IT) security professionals should conduct a thorough assessment of the logical security of the infrastructure. Although a description of that assessment is beyond the scope of this entry, at a minimum, the following questions should be answered:

- What are the back-up procedures? For example, how often is the development environment, source code, and compiled code backed up? Where are they stored? Who has access to the back-up media?
- Have any tapes been restored to validate the back-up process?
- Is there a business continuity and disaster recovery plan to detail how restoration and development activities will continue in the event of an incident?
- How are logical access controls managed for the source code, executable build systems, and test systems? Who approves the access list? When was it last reviewed?
- Have unnecessary services been turned off on the servers and workstations? Are updates and security patches regularly applied?
- How do the developers authenticate to the servers? Is traffic encrypted? Are clear text protocols used (such as Telnet)? If developers are using X Windows, has the configuration been reviewed?
- Are the developers and the development infrastructure segmented from the corporate network? A great
way to add an additional layer of logical security is to segment the development environment from the rest of the company via a firewall with well-designed policies permitting only the required traffic.

- Are the access logs to the servers, firewalls or routers (if applicable), and workstations reviewed for security events and investigated when required?

Now that the environment where the software will be developed has a secure baseline, we can focus our attention on the foundation of the development activities themselves. As part of that foundation, developers should have a minimum baseline of knowledge or awareness of security vulnerabilities, coding best practices, and industry trends and best practices.

There are numerous resources available, including Web sites, magazines devoted to information security, training programs, and organizations that offer specialized classes and seminars. Several security training organizations have offered classes in the past, magazines have published excellent articles on building security into the SDLC, and several excellent books have been published detailing specific vulnerabilities and how to avoid them, as well as how to develop a methodology to improve the reliability and security of software systems. Finally, numerous Web sites, online articles, and Web-based seminars have offered free, relevant, and very timely advice on how to produce secure software.

As a further reason to help encourage the development of secure code, senior management may wish to consider rewarding developers who reduce the number of security vulnerabilities within their code, or perhaps rewarding quality assurance personnel who discover vulnerabilities prior to deployment. In any event, it is important to ensure that all team members are educated and aware of the resources that are available to them, and have the commitment from management to allow them the time and resources to learn.

The education process should not be a one-time effort, but instead built into the overall SDLC to ensure that each team member’s skills are continually honed and enhanced. Additionally, new attack vectors (where and how attacks originate) and new vulnerabilities are regularly announced. Keeping abreast of specific language, software development kits (SDKs), and development environment vulnerabilities can be accomplished through vendor training, subscriptions to vulnerability announcement mailing lists, and subscription services, as well as through participation in industry and user groups.

Vulnerabilities are many and diverse. Structured Query Language and XML code injection, buffer overruns, race conditions, improper storage of cryptographic keys, format string errors, cross site scripting, and poor usability leading to the user disabling some security features are just a few of the vulnerabilities that must be mitigated in today’s code. If designers and coders are not aware of the range of vulnerabilities, they may not be able to avoid them. If quality assurance personnel are not aware of the different types of vulnerabilities, they cannot test for them and alert the coding team. Continuous awareness and training sessions for all team members must be a requirement and part of each associate’s annual review process.

Conceptual Design

After the organization has a basic security awareness foundation, it is time to form the team to begin the first step, which is typically conceptual design. As I re-read this entry, I noted that I have said that each SDLC phase was the “most important” from a security perspective. Let us consider the conceptual phase that really sets expectations for the overall functionality of the application. Security personnel at this phase should be providing guidance based on known threats, vulnerabilities, risks, and available and potential controls. Although not necessarily driving the end result, security input early on can help define what can and cannot be done. As an example, and I am not making this up, an organization wanted to develop an application that required real-time access to a critical system on our company’s intranet for Internet users. Although it could have been done securely with the addition of numerous and costly controls, designing a tiered DMZ infrastructure allowed the development team to implement multiple other features, delighting the sales and marketing team and making the IT security organization even happier.

Technical and Functional Requirements

The next step in the SDLC is the formulation of the functional and technical requirements. As noted previously, these are sometimes completed in parallel or combined. For the sake of this entry, we will discuss the functional and technical requirements as a single phase. As a very simple example, consider the functional requirement that the application “must read input on a text file output by another program” and a technical requirement that the application “must read standard ASCII comma delimited text, fields up to 256 characters, with a record size limited only by the storage capacity.” What happens when the format is not comma delimited, or when the fields have 50,000 characters? We typically do not put the negative cases in the requirements documents, but that is how we typically get into trouble with buffer overflows, unchecked inputs, and so on. Defining and understanding the entire range of inputs (not just what is expected) and defining the
requirements for responding to all input, whether expected or not, are paramount to system security.

During the technical and functional requirement phases, it is imperative that the security consultant provides inputs and direction regarding the security requirements. Although it is unwieldy to add the requirement to check for buffer overflows, unchecked inputs, and so on at every input requirement, it is necessary to capture the overall requirement that all input will be checked and validated prior to processing. In addition, there will likely be several key areas that will be detailed in this requirements section that will need to be incorporated into the application.

Depending on the system under development, there are likely numerous privacy requirements that must be incorporated into the final system. The source of the privacy requirements may come from any number of sources, including:

- Health Insurance Portability and Accountability Act (HIPAA) of 1996
- Gramm–Leach–Bliley Act of 1999
- European Privacy Directive
- Canadian Privacy Act
- The development organization and end customer’s privacy standards.

The privacy requirements will typically drive how information is stored, how it must be transmitted, back-up requirements (such as requiring encryption), how long data can be retained, how and with whom it may be shared, and how it must be destroyed. Finally, privacy requirements will drive the business continuity and security requirements that are discussed next.

In addition to privacy requirements, there will likely be disaster recovery and business continuity requirements that will need to be incorporated into the application. If the system is going to support a critical business process or perhaps be one, failover, redundancy, and back-up features will likely be included in the overall requirements. Specifications as to the types of back-ups, transaction logs, parameters of system heartbeats to support hot-swappable capabilities, and perhaps how the system manages the fail-over process will be part of the requirements. As part of the requirements phase, security consultants must be tasked with identifying the relevant regulations that will influence the application and provide input based on those regulations and industry best practices. To accomplish that, an understanding of the customer base, including where they will use the application and what it will be used for, will be needed so as to incorporate the applicable requirements for that region or industry.

The security requirements will also influence how the system traverses the remainder of the SDLC. There will be many security requirements that will be part of the system. Validating all input, authentication, encryption of data in transit and rest, and authorization must be addressed. Roles and corresponding responsibilities must be defined and be flexible and granular enough to ensure that “least privilege” concepts are met while not interfering with the day-to-day activities of the system.

One of the most comprehensive efforts to identify the requirements from a security perspective is the development of a threat and vulnerability matrix, or an attack vector. Through this exercise, commonly undertaken as part of a risk assessment, comes the understanding of the threats, vulnerabilities, and computed risks that a software system will face upon deployment. Vulnerabilities of the host operating system, auxiliary systems, threats to industries where the application may be deployed, its target (and potential) audience, and mitigating controls that may be placed into effect alongside the system are examples of inputs to the threat and vulnerability matrix. By developing an attack vector of what segments or functions of the system are likely to be vulnerable, special attention can be paid to those areas to ensure a strong resiliency to attack. It must be noted that threats, vulnerabilities, and controls are continually changing, and it would be negligent to ensure that the software is resilient to attack only at the areas identified in the threat vectors. The attack vector approach should only be used to ensure that the segments most likely to be attacked have sufficient controls and that all functions of the application enjoy a similar level of protection.

One can also consider conducting a risk assessment of the proposed system. Knowing that a commonly-accepted definition of the value of risk is Risk (System) = (Threats × Vulnerabilities)/Controls, we can compute the value of risk, and then, as the project moves from the design phase to coding and implementation phases, the value of risk can be continuously measured and monitored, and reduced as necessary to achieve a sufficiently low level. Noting that the risk equation cited earlier is defined as a function, we can compute the risk of any or all components of the system depending on our area of interest or review.

Significant events must be logged. Questions to be answered include what is logged, the location to which it is logged, what happens when the log fills up (i.e., does the system halt, or does it overwrite the oldest log data?), whether the logs are stored locally or remotely, and whether they can be centrally monitored. Access to the logs and control of the logging configuration are equally important, as either could afford a malicious user the opportunity to hide the tracks of an attack. It is the responsibility of the security consultant to ensure that minimum standards of logging (as well as other security-sensitive areas) as identified in any corporate policies are incorporated into the system’s requirements.
Databases require particular attention, as they are typically the stores of the data processed by systems. Ensuring that default and system accounts are disabled unless the functionality is required, and then changing passwords of required system accounts, would be ideal requirements. Setting strong passwords on system accounts so they are resilient against long-term, brute force attempts should be a requirement as well. Requirements should include encrypting at the database level, defining authorization for read, write, and deletions, as well as how the database is to be accessed through the software system, that is, through the databases console or other third-party applications.

System Design

In the design phase, the functional and technical requirements are used to architect a system at a high level by decomposing it into functions, modules, libraries, etc. Participants in the design phase should have a thorough understanding of the hardware requirements (if applicable) of the system and should develop a design that is sufficiently robust to withstand attack when implemented on noncompliant hardware with drivers that were not validated or on operating systems that have never been updated or patched. In many commercial software development projects, it is impossible to predict the target platform software, operating system, other applications or services on the system, and so on. Systems that do not make assumptions about trusting the operating system, hardware, and other applications will fare better than those that blindly accept all inputs or transactions. Just like in real life, systems should trust, but verify.

At the design phase, the developers should be aware of the available controls and should be designing the system to maximize their use, while including additional controls to mitigate all threats and vulnerabilities previously identified during the threat and vulnerability discovery or risk assessment phases. Finally, the designers should include built-in mechanisms that regularly check for updates to the system and are able to receive and install those updates regularly and easily.

Coding

When the coding phase is initiated, a solid set of requirements should exist that highlights the technical and functional requirements of the system. These should include security requirements. The coding personnel should know they have the additional responsibility of implementing features, functions, and attributes of the system with security functionality in mind, even when it is not explicitly defined in the requirements. Care should be taken to review requirements with the marketing organization, sales group, end users, or end customer when the organization that is responsible for coding has not been part of the entire SDLC.

Development efforts should utilize a source code management system that is adequately secured to protect source code assets from unauthorized access, disclosure, modification, or deletion. User account management, logging, and auditing should be carefully managed and regularly reviewed to ensure that personnel have access only to the data they need for their work and that they are authorized to access. Change control and configuration management are two important programs that support security requirements and are likely supported by features within the source code management system.

The coding phase introduces a number of areas that must be considered, including the complexity of the system, the application development language, the integrated development environment (IDE), the use of SDKs, and use of code libraries. The use of code libraries and SDKs introduces new challenges to the SDLC, as the source code may not always be available to the development team for review, and usually only provides the defined interfaces, such as how to call the application and what each function does. Its resiliency to a buffer overflow attack may not be known and may need to be tested in a black-box fashion detailed later in this entry.

Although the number of tools available for Web-based applications exceeds that available for traditional executable applications, there are many tools that integrate with IDEs to provide immediate feedback when they suspect potential security coding vulnerabilities. Just as a word processor highlights misspelled words as the user types, applications are available to highlight potential errors in the code that could be compromised. Although this solution should not be considered the sole control during the coding process, it is a very strong and successful approach. Doing a Web-based search for application coding vulnerability scanners will highlight some of the tools that are available commercially or through open source efforts. Although some are significantly better than others, cost, vendor preference, programming language, and IDE are factors that will drive the decision-making process. Many of these products have complementary products that provide similar testing features on the compiled or Web-enabled applications after they are installed. Typically, although not a requirement, IDE-based programs serve the needs of developers, whereas the tools used to scan executables or Web-based applications are used by auditors, assessors, and quality assurance personnel.

During the coding phase, code reviews should be conducted to provide peer review and feedback. The subject of many books and articles, code reviews are simply an opportunity for software coders to share their code with other coders to solicit their feedback,
comments, and insights. Typically not focusing solely on security vulnerabilities, a code review serves to identify inefficiencies, areas of potential code re-use, logic errors, and suggestions for cleaner or more robust code. For critical interfaces and processes, a larger team may be deployed to include other members of the SDLC team, such as designers and quality assurance personnel.

“Formal methods” is a software engineering process in which mathematical and logical proofs are used to “prove” that the software is correct, or does what the requirements specify that it should do. The formal-methods approach provides additional insights for validating software, although it is typically time and resource intensive, as it is often quite a challenging effort with only a few automated tools to provide assistance. Finally, the formal-methods approach can be used to prove that code handles inputs as intended and properly rejects code that is incorrectly formatted or is invalid.

“Secure by default” is a term we hear quite often these days; it refers to the initial values of the various settings, parameters, and configurations. For example, consider a program that advertises that it securely uploads files to a remote server on a nightly basis over the Internet. Unless the operator knows that it is possible to enable the “secure copy” option, the program may utilize the traditional file transport protocol that sends the account information and data in clear text. With the secure copy option enabled, the transfer is significantly more secure. “Secure by default” initially enables the security features of the system and thus increases the overall security. End users must indicate that they do not want the default level of security by disabling or reducing the security controls.

Finally, the code must be documented. Although one can argue that secure code can be developed without documentation, best practices require that source code be commented and that sufficient documentation exists to detail how the code was developed in support of the requirements. In the event of vulnerability announcements in time to come, commented code can support reviews and investigations as to which code may need to be redeveloped.

A common security error that originates in the coding phase is the use of test data that is real customer data. Although using data that is valid and representative of real-word situations, it is important to note that, in many instances, using customer data for coding and testing procedures may be in violation of federal regulations stipulating that data must be protected. There are several ways to accomplish testing without using such data, including creating entirely random data, manually populating a test database, or using algorithms like as one-way hashes to mask the data used in testing. Creating artificial data can leave testers without the invalid or unchecked data that may often exist in real-life data. The SDLC team should utilize a dataset that contains both sufficient valid and invalid data to test exception cases that will inevitably be encountered in operation.

### System and Unit Test

The test phase should be the last line of defense for discovering security vulnerabilities, not the front line. Using the test phase to catch vulnerabilities in the code base not only increases costs to correct the code, but detracts from the other responsibilities of the quality assurance personnel who are also reviewing documentation, installation, operation, interfaces with other systems and processes, as well as the logic of the application.

As noted during the previously discussed coding phase, there are several applications that are available to review and test the code for not only logic errors, but for security vulnerabilities as well. If the quality assurance personnel have been involved with the project from the earliest stages, test plans, test cases, expected results, and areas of concern should have been identified and documented. Code utilized as part of an SDK or that is received as precompiled will have to be reviewed as well. These reviews can use black-box testing, a term that is applied to testing code when you have no insight into the source code and can only supply different inputs (some within the interface parameters and some that are not), to ensure that the output is as expected.

Finally, there are many applications available to quality assurance personnel that provide support in automated testing. Applications that can learn expected responses, offer scripting, accept various forms of input, and automatically capture and flag suspect results can be utilized to reduce the time and resources required for testing, or more importantly, to allow the testers to investigate suspect and questionable results.

### Deployment

The SDLC continues after the software has been designed and coded, as efforts begin to package, ship, deploy, and implement the software. Depending upon whether the software is a customized software solution or a commercial off-the-shelf solution (COTS), the involvement of the vendor will vary. During the initial deployment, quality assurance and design personnel should be closely supporting the help desks to provide guidance and, most importantly, to identify trends and patterns that may indicate vulnerability. In addition, Web-casts, alerts to customers, awareness training for employees, and so on, may be useful mechanisms for informing and educating users about the secure operation and management of the system. Finally, the system’s documentation may require updates and
clarifications based on feedback from the help desk to ensure clarity and understanding of the security features.

The installation package is created to facilitate the installation of the software. Proper testing should be performed to ensure that the installation doesn’t introduce additional vulnerabilities (such as network-based installation packages that may introduce specialized services to support the installation); the latest documentation should be provided to the customers as well. Finally, customers should be made aware of mechanisms for receiving updated software packages and documentation as they become available.

Depending upon contractual requirements for customized software development, as the system moves into deployment, the release version of the source code may be transferred into “escrow” or may be transferred to the procuring organization itself. Although the escrow contract may dictate how the software is to be transferred and stored, appropriate measures must be taken to protect the data while in storage and transit, while still providing access to authorized users. The storage and management of cryptographic keys will need to be planned and agreed upon by the development firm, the end customer organization, and the escrow organization (when appropriate).

Software System Maintenance

Once the software system begins to ship, the maintenance mode typically begins. Vendors usually offer several years of support for each release for COTS-based packages, whereas the support for customized software is generally dictated by contractual terms. In any event, the vendor will typically receive input from:

- Customers who have uncovered potential security vulnerabilities
- Security research firms who are continually reviewing and dissecting applications and operating systems of all types
- Vulnerability announcements from the manufacturers of the IDEs, SDKs, and the compilers and language developers
- Continued quality assurance testing efforts that may uncover existing vulnerabilities while testing new features and updates

It will be critical to the organization’s reputation and customer service to be able to accept and acknowledge vulnerability information and to be able to validate that information before issuing updates that mitigate the vulnerability in a reasonable time. There are a number of competing factors regarding disclosure. Some believe in “full disclosure,” which is the release of vulnerability information as soon as it is made available. The argument for full disclosure says “If I find vulnerability in a software package, everybody should know about it to provide an opportunity to implement additional controls.” The argument against full disclosure is that as those with malicious intent become aware of the vulnerability, the clock begins to tick for the development of malware, viruses, and Trojans that will exploit that vulnerability. As a compromise, de facto standards have emerged that highlight recommended timelines, communications, and interactions between the discoverer of the vulnerability and the manufacturer of the vulnerability. COTS applications that must run on various platforms and multiple operating system versions may require lengthier timeframes (sometimes 30 days or more) to include regression testing, documentation, and packaging, whereas open sourced applications (and some commercial applications as well) have taken just a few hours to release a patch.

**Decommissioning**

Although the decommissioning phase can be as simple as clicking on “Uninstall,” the removal of associated data and other configuration information is of the most concern. For example, if the application is uninstalled, then application data (which can be contained in anything from text files to relational databases) as well as configuration information (such as cryptographic keys and stored user names and passwords) must be deleted. Additionally, any adjunct services that were installed must be removed unless they are required by other applications. This is often a tricky task as the user must guess if any other installed applications require that particular service. Secured or not, it is not prudent to leave a service running when it is no longer needed.

During decommissioning or uninstalling, the user must be provided with options for what should be done with application data, cryptographic data, or user account information. If the user requests deletion of the data, then the user should be informed that data is not truly “deleted” and may be easily recovered with readily available tools. The uninstall function should provide recommendations on how to securely delete the data if it is considered sensitive. If application data is to be retained for use or for back-up purposes in time to come, appropriate security controls should be instituted to protect the data.

**CONCLUSION**

With security research firms paying a bounty to receive previously unannounced vulnerability information to boost the awareness of their firms and their credibility, and with malicious individuals paying a bounty to be...
the first to generate exploit code, it is critical for software development firms to incorporate timely and efficient mechanisms for managing security vulnerabilities from discovery through delivering an update. Freelancers, white-hat, gray-hat, and black-hat hackers have devoted careers to reviewing, disassembling, reverse engineering, and trying every combination and permutation of inputs and configurations in an attempt to find the one scenario where the system crashes, releases some private information in an error message, or allows some arbitrary code to run.

Software development is a customized process with many equally valid options for how to reach the end state. Programming languages, styles, environments, platforms, and designing and coding experience are all variables that will ultimately shape the end result, including how it operates, how it interfaces with other components, and how it works on various hardware and system platforms.

Through the development and use of a continually-updated policy framework, the development team will have the basic information of how software must be developed in the organization. Equally important is the continual training and awareness of the entire team of existing threats, vulnerabilities, industry best practices, and most importantly, regulations, that they must be aware of and compliant with. It is important to note that many tasks in this entry, particularly those of developing a strong policy framework and awareness, must be continually updated. Vulnerabilities and threats continue to change. New ones are added, and older ones are mitigated regularly. Having a program in place to develop software that is resilient in the face of vulnerabilities of the present as well as the time to come will allow a company to survive. Having a program in place to update its software in a timely manner when security issues arise will allow a company to build customer confidence and thrive.

The delivery of a secure software package is the goal of every development organization. Perhaps a realistic goal is to develop software in which the known security vulnerabilities are mitigated, or have sufficient controls in place, and the discovered vulnerabilities are managed in a timely and professional manner.