Software Security: Being Secure in an Insecure World

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA

Introduction

Door locks, gated communities, guard dogs, access cards, and identification badges are all testaments to a physically insecure world. Likewise, the need for similar protective mechanisms is no less significant in that part of our information age which is not physical. Our digital world is every bit as insecure as our physical one. The information age, in fact, is an extension of the industrial age, characterized by the focus on production of physical goods. Today, most manufacturing efforts are augmented and in some cases managed by components of the information age.

Ubiquitous software is a characteristic of the information age. It has become a crucial component of day to day living for most of the world, and it heavily influences the world's very social and economic fabric. Software is used today for communications, production, financial transactions, transportation, and utilities to name just a few of its varied and countless uses. With software, technical solutions to business problems are possible. And, with software, we can all be connected.

And while a lot of effort goes into designing, developing and deploying software, with internetworking connectivity (Internet), making the world smaller, there is little to nothing being done to make it any more secure. Disney's famous tune that asserts incessantly to visitors at the Disney theme parks that, "It's a *small* world, after all" can accurately be sung today with the words "It's an *insecure* world, after all."

(ISC)$^{2®}$'s whitepaper, *The Need for Secure Software* addresses the "Why" of securing software. It delves into the drivers of software assurance, the importance of data security, and covers the policy, process, and people aspects of software assurance. (ISC)^{2}'s whitepaper, *Software Assurance: A Kaleidoscope of Perspectives* addresses the "What" of software assurance in terms of the varied perspectives that need to be considered when building secure software. This whitepaper on *Being Secure in an Insecure World* will address the "How-Tos" of designing, developing, and deploying secure software.

SwAconomics – Insecure Software Cost

Software is merely sets of instructions given to computers to be followed as instructed. These instructions are designed by humans and hence software is only as strong or as weak as the designer. As somebody once said, expecting a computer to think is akin to expecting a submarine to swim. Current trends of software insecurity, as reported in the press highlight the unfortunate reality that software today is inherently not as secure as it should be. In fact, the industry is still maturing in the arena of software security and has a lot of ground to catch up. The lack of globally enforceable regulations and legal jurisprudence only exacerbate the situation.

David Rice, former cryptographer for the NSA and Navy, author of *Geekonomics: The Real Cost of Insecure Software*, approximates, as reported on Forbes.com[a], that the total economic cost of security flaws in software is around US$180 billion a year. While the economics of software assurance (SwAconomics) can be extrapolated from the aggregate amount of fines levied on organizations that have experienced a breach due to insecure software, that still doesn't provide a complete view of the cost of insecure software. The real cost, not as neatly quantifiable, is the extent of reputational damage and loss of customer trust. For example, arguably the most noted U.S. customer data breach incident occurred at TJX stores where the recovery cost from that particular breach is estimated to be approximately US$216 million. But the more significant problem in the long run will be gaining back the confidence of the customers, which may prove difficult if not altogether impossible.

Unambiguous client requirements and solid design and development can result in quality software. It is important, however, to recognize that quality software does not always imply secure software. This is evident from the fact that there is some extremely useful and productivity-enhancing software currently on the market, but software that has nevertheless been exploited from a security standpoint or has the potential to be exploited. The inverse however is true. Secure software implies quality software; quality in terms of confidentiality (not disclosing information), integrity (not allowing unintended alterations), and availability (reliability).

The Blame Game

When insecure software is exploited or has the potential to be exploited, who is to be blamed? Is it the product manager who did not factor in the necessary security controls when translating business requirements into functional requirements? Is it the project manager who did not factor in adequate time and resources for ensuring that security design and architecture review was adequately performed? Is it the developer who did not write secure code? The tester who did not validate security functionality? Or is it the operations personnel who are responsible to maintain security? Or, taking things a step higher, maybe it's really the executive responsible for the delivery of the software. In some sense, is the company as a whole responsible?

Former Burton Group Vice President, and founder of Security Curve, Diana Kelley expresses in her acclaimed paper, *Application Security: Everybody's Problem*[b] that software (application) security is the responsibility of all the stakeholders that are influencers in the software development life cycle (SDLC). She goes on to state that enterprises that understand how to create more secure applications can benefit from greater efficiencies in the development process, thereby reducing the need for post-deployment security software whose function is to protect and patch insecure software.

In other words, focusing after the fact on who's to blame does not properly address the issue of insecure software. The "blame game" keeps the organization in a circuitous cycle of the inefficient and reactive "patch and release" *modus operandi*.

Secure Practices in the SDLC

While some of the insecurity in software could be the result of the technology chosen, it is important to note that software products are predominantly insecure due to two other elements – people and processes. Any software is the result of a confluence of people, process, and technology. Secure software is the result of educated and informed people implementing hack-resilient processes using inherently secure technologies to provide solutions to a business need.

Security is a process; from requirements to release it is to be woven into the SDLC. Software products built today are primarily focused on business functionality and features. Even though the SDLC may cover quality-control planning and testing, seldom does it incorporate security holistically. From requirements to release, there are a lack of adequate security controls that need to be built into the SDLC, and security requirements are in many cases non-existent. Use cases are not complemented with their inverse misuse cases. Threat modeling, when present, is often performed by a trained security professional instead of members from the development team and when subsequent changes are made by the developers, they are rarely retrofitted into the threat model. Developers are driven to deliver functionality with deadline and scope constraints, pushing the writing of secure code to the sidelines. Testers are often inadequately trained to look for security vulnerabilities. Finally, when developed and released to the public as commercial-off-the-shelf (COTS) software, or deployed into production, as in the case of internal business software, software that does not factor in security through its life cycle is often rife with vulnerabilities that practically implore an attacker to exploit them.

The following sections cover the various aspects of secure practices through the different phases of the SDLC.

1. Requirements Gathering

Not incorporating the core tenets of security (confidentiality, integrity, availability, authentication, authorization, and auditing) in the requirements phase of a software development project will inevitably result in software that is insecure. Since software, like anything else that goes through a manufacturing process, is designed and developed to a blueprint, it is of paramount importance that security requirements are determined alongside the functional and business requirements. In other words, security requirements need to be an integral part of the blueprint itself, as shown in table 1.

Table 1. Tools and Process Recommendations for the Requirements Gathering Phase of the SDLC to Build Secure Software.

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Requirements Gathering	Business Partner Engagement Identify Policies and Standards Identify Regulatory, Compliance, and Privacy Requirements Develop CIA* Objectives Develop Procurement Requirements Perform Preliminary Risk Assessment	Business Partner Questionnaire Policy/Standards Checklist Local and International Checklists CIA Questionnaire Data Classification Procurement Checklist Rapid Risk Triage / Prototype or Questionnaire

* Confidentiality, Integrity, and Availability

a. Engage the Business Partner or Client

In addition to ensuring that the software developed will meet the business or client functionality requirements, engaging the business partner during the requirements-gathering stage to address security aspects will aid in the partner's understanding of the risk, and assist in eliciting the protection needs of the software. Using a questionnaire or checklist in language that the business partner understands (without going too deep into technical or security jargon) works well in uncovering security requirements.

b. Identify Applicable Policies and Standards

It is critically important that software developed is done by following established policies and standards and is compliant with audit requirements. For example, if your authentication standard lists the need to have multi-factor authentication, then the software you build should be compliant to that standard.

c. Identify Applicable Regulatory, Compliance, and Privacy Requirements

It is important that software requirements take into consideration the regulatory (legal), compliance, and privacy requirements. These considerations should not only be local but also international.

d. Develop Confidentiality, Integrity, and Availability Objectives

During the requirements definition phase, it is essential to develop the Confidentiality, Integrity, and Availability (CIA) objectives of the software. Is the data or information open for viewing by all or should it be restricted (confidentiality requirement)? What are the factors that allow for authorized alterations, and who is allowed to make them (integrity requirement)? What is the accessibility of the software and what is the allowable downtime (availability requirement)? In addition to the CIA requirements, it is also necessary to consider the software Authentication aspect (proving of claims and identities), Authorization aspect (rights of the requestor), and Auditing aspect (accountability or building historical evidence).

Data classification is a proven methodology in assisting with the determination of the CIA goals and objectives. It can also help in prioritizing and determining the appropriate level of security controls to be incorporated into the software. Sun Microsystems's whitepaper, *Best practices in data classification for information lifecycle management*[c] delves into the best practices in data classification and addresses the need, process, and benefits of it.

e. Develop Procurement Requirements

If software is to be bought, rather than built in-house, developing the procurement requirements is important. Care must be taken to ensure that new levels of threat or risk are not introduced into the existing environment in which the software will run. A clear understanding of the current environment is necessary and engaging the architecture, networking, engineering, operations, and security team along with the procurement group aids in this objective.

f. Perform Preliminary Risk Assessment

A preliminary risk assessment is necessary to determine the fundamental security necessities of the software. This risk assessment should not be onerous, but just thorough enough to get a picture of the risk that the software will introduce. A questionnaire uncovering the essential requirements of CIA and a rapid risk triaging[d] model are both useful methodologies for risk assessment.

Table 2. Tools and Process Recommendations for the Design Phase of the SDLC to Build Secure Software

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Design	Misuse Case Modeling Security Design and Architecture Review Threat and Risk Modeling Security Requirements and Test Cases Generation	Requirements Traceability Matrix Security Plan Threat Model Security Test Cases Template

2. Design

It's a given that if the core tenets of security are not included as requirements, then the software when designed is probably not going to have them. But even in cases where security requirements are determined, they often run the risk of being dropped from the feature specifications or being lost in translation due to the constraints of time and budget, and/or a lack of understanding of their importance by the business or client. Project managers should plan and allow for time and budget to ensure security requirements are not ignored.

Applying to software security the 80-20 rule (Pareto Principle) which states that 80% of the effects come from 20% of the causes, it's no surprise that 80% of the software defects arise from 20% of the design flaws. Addressing the 20% of design flaws during design can mitigate the exposure factor considerably.

a. Modeling Misuse Cases

From the vantage point of security it's not only important that the functionality of the software is depicted in use cases, but it's critical that the inverse of the use cases (misuse cases) be modeled to understand and address the security aspects of the software. Use of a requirements traceability matrix will assist in tracking the misuse cases to the functionality of the software.

b. Conduct Security Design and Architecture Reviews

It's important to recognize that, in most software development projects, time and budget are fixed, and the introduction of security requirements are generally not well received by software development teams. The best place to introduce the "security" design and architecture review is when the teams are engaged in the "functional" design and architecture review of the software. When conducting a security review, the assurance requirements of the software should be considered bearing in mind the cost and time constraints. Generating a security plan from the review is a good start for documenting the security design and using it as a check-and-balance guide during and after development.

c. Perform Threat and Risk Modeling

Threat modeling includes determining the attack surface of the software by examining its functionality for trust boundaries, entry points, data flow, and exit points. It is to be performed only after the functionality requirements are complete, so that the threat model is based on the functionality of the software. Threat modeling is useful for ensuring that the design complements the security objectives, making trade-off and prioritization-of-effort decisions, besides reducing the risk of security issues during development and operations. Risk modeling of software can be accomplished by ranking the threats as they pertain to your organization's business objectives, compliance and regulatory requirements and security exposures.

d. Security Requirements and Test Cases Generation

Modeling of misuse cases, security design and architecture reviews, and threat and risk modeling can all be used to generate the security requirements that the developer should write code for, and to determine the security test cases that should be executed during testing. Using a scenario-based security testing template is effective in ensuring that the bare minimal security test cases are performed in every software development effort, as well as saving time in generating test cases that are essential.

Table 3. Tools and Process Recommendations for the Development Phase of the SDLC to Build Secure Software.

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Development	Writing Secure Code Security Code Review Security Documentation	Security Checklist Code Scanners
Testing	Security Testing Redo Risk Assessment	Security Test Cases

3. Development/Testing

The software written should be secure by design, secure in development, and secure by default. Defense in depth and least privilege should be to the forefront when it comes to writing secure code. Layered defense should be built into the software to avoid any one single point of failure.

a. Writing Secure Code

Contrary to popular opinion that software security is all about writing secure code, and although it is a critical step in SDLC, secure code writing is only one of the various steps necessary to ensure security in software. Software developed should at the bare minimum be written to mitigate the common and prevalent threats in the industry, such as overflow attacks, injection attacks, scripting attacks, and remote code execution attacks to name a few. Using a security checklist can ensure that minimum security baselines pertaining to writing secure code are covered.

b. Security Code Review

Automated or manual code reviews should be performed during the development phase of the project to make sure vulnerabilities in the code are discovered prior to release or deployment. When code review is automated, it's important to bear in mind that it should be done in addition to manual reviews, and not in lieu of them. Control checks by a human should still take place. Special attention should be given to false positives and false negatives of the automated code reviews.

c. Security Documentation

The security plan generated during the design phase of the project must be revisited and adjusted if necessary. It is imperative that any change to the security plan be made only in those situations where achieving the security objective is improbable or infeasible due to extraneous factors beyond the scope of the project.

d. Security Testing

Critically important in the life cycle of a secure software development project is that security testing be performed in addition to functionality testing. Educating the testers to become software security testers not only boosts the technical aptitude of the quality assurance organization, but also results in software products that are more secure. Capturing the security testing requirements in the design phase and executing them in the testing phase are vitally important.

e. Redo Risk Assessment

Post development, a risk assessment will help identify the risks that have been mitigated and the ones that still need to be addressed. This will give the SDLC project stakeholders the ability to decide on the acceptable risk level and whether or not to release/deploy the software.

Table 4. Tools and Process Recommendations for the Deployment Phase of the SDLC to Build Secure Software.

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Deployment	Secure Installation Vulnerability Assessment and Penetration Testing Security Certification and Accreditation (C&A) Risk Adjustments	Environment Configuration Document Vulnerability Assessment Plan Penetration Testing Procedures C&A Workflow

4. Deployment

All efforts to design and develop secure software are rendered futile if the software is not securely deployed.

a. Secure Installation

Software development that does not factor in least privilege often produces software that runs without any issues in a lax development environment. But when deployed to a more tightly controlled and secure production environment, such software fails. In such situations, administrators often are forced to reduce the tight control or increase the rights with which the software will run, both of which are forms of insecure installation. Software should therefore be tested in environments simulating the production environment, be it system integration testing environment or the user-acceptance testing environment. Installation should never reduce the security configuration of the environment, thereby increasing the attack surface of the software and the overall risk to the environment.

b. Vulnerability Assessment and Penetration Testing

Vulnerability assessments (VA) and penetration testing (PT) should be performed to determine the risk and attest to the strength of the software after it has been deployed. Although vulnerability assessments and penetration testing are used synonymously by many, they are not the same. Vulnerability assessment is a process of identifying known weaknesses of software. Penetration testing on the other hand is testing the security of the software, simulating a malicious attacker. A part of vulnerability assessment can be penetration testing.

c. Security Certification and Accreditation (C&A)

The National Institute of Standards and Technology (NIST®) describes security certification as the process that ensures controls are effectively implemented through established verification techniques and procedures, giving organization officials confidence that the appropriate safeguards and countermeasures are in place as means of protection. This in essence is a formal methodology and has the same output as that of a vulnerability assessment – the weakness of software. Accreditation on the other hand is the provisioning of the necessary security authorization by a senior organization official to process, store, or transmit information. This is based on the verified effectiveness of security controls to some agreed-upon level of assurance and an identified residual risk to agency assets or operations. As deemed necessary, both certification and accreditation should be performed before deploying software.

d. Risk Adjustments

Vulnerability assessments, penetration testing, certification and accreditation exercises will all provide insight into the residual risk introduced by software. Necessary adjustments of the risk profile should be made. Contingency plans and exceptions should be generated should the residual risk be above the acceptable threshold.

Table 5. Tools and Process Recommendations for the Maintenance Phase of the SDLC to Build Secure Software.

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Maintenance	Change and Configuration Control Recertification & Reaccreditation Incident Handling Auditing Continuous Monitoring	Change Control Process C&A Workflow Incident Management Plan Audit Review Plan Monitoring Procedures

5. Maintenance

Confucius said, "The superior man, when resting in safety, does not forget that danger may come. When in state of security he does not forget the possibility of ruin. When all is orderly, he does not forget disorder may come. Thus his person is not endangered and his states and all their clans are preserved." Applying this wisdom, it is easy to see that when it comes to software security, not only should software be designed, developed, and deployed securely, but it should also be operationally secure and should maintain the level of security as intended.

a. Change Control and Configuration Control

Proper change control and configuration control should be in place to ensure that only approved changes are made to the code base of the software. Versioning of software with auditable check-in and check-out procedures is essential. Direct access to production code should be prevented and be on a "need to know" basis with explicit authorization and controlled scrutiny. In cases when controlled and scrutinized access to production code is granted, it should be for primarily troubleshooting purposes (if the issue cannot be recreated in the testing environments) and no change should be allowed in the production environment directly. Changes that need to be made should be made in the development environment, tested thoroughly in the testing environment and then migrated to the production environment during the approved change windows.

b. Recertification and Reaccreditation

Upon any change, should there be a need to re-certify and re-accredit the software, necessary processes should be established and followed to get certification and accreditation.

c. Incident Handling

An Incident Management plan is essential and should be established. The plan should include the reporting procedures, escalation paths, assurance of anonymity (if needed), abuse reporting emails, hotlines, and other mechanisms to encourage individuals to report the issue, when an incident occurs or is suspected. Reported incidents should be managed effectively by controlling information to only the needed parties.

d. Auditing

Auditing refers to the logging of necessary information that can be used to build historical evidences of changes. What to log and how long to retain logs is based on the criticality of the changes being made, authorized or unauthorized and the records retention and information management guidelines of your organization. All administrative functionality and business-critical activities should be logged. The date, time, and user or process that made the changes should be logged. Special care should be taken to ensure that the log files and records are secured as well, as they may contain sensitive information. Secure software records logs by default. Audit logs can also be used as detective controls in the event of an incident.

e. Continuous Monitoring

Through recurrent testing and assessment, security controls built into the software are validated for their effectiveness. Operations personnel should monitor deployed software to ensure that software security is not affected or reduced over time. Any unintended behavior of the software should be reported to the software development team and an investigation to determine cause should be undertaken.

Table 6. Tools and Process Recommendations for the Disposal Phase of the SDLC to Build Secure Software.

SDLC Phase	Security Control (What to do?)	Recommendation – Tools/Processes (How-Tos)
Disposal	Secure Archiving Data Sanitization Secure Disposal	Records Management Policy Data Sanitization and Disposal procedures

6. Disposal

Just as it is important to build secure software, it is equally important to securely dispose of software once its usefulness and regulatory obligations have been met. Disposal implies not just data sanitization and destruction, but also archiving.

a. Secure Data Archiving and Sanitization

It is vital to ensure that when software is disposed, information that would be needed at a later timeframe is archived for future retrieval by secure means. Archived information and software should be treated and protected with the same control measures as one would employ with confidential information. The archival requirements should be in accordance with the organization's record management policy or standards. Sanitization of data refers to overwriting or destruction of information no longer necessary to be preserved.

b. Secure Disposal

In cases where data and software is highly sensitive and no longer necessary, it must be physically destroyed. If the software was used to store data in offline media, care should be taken to destroy the storage media and if necessary the software as well.

A New Culture: Software Lifecycle Influencers with a Security Mindset

When it comes to building secure software, people can be the strongest force or the weakest link. A primary shift is necessary in the mindset of all of the stakeholders in the SDLC process, one that makes security second nature in the software they are responsible for building. For such a shift to happen, these stakeholders need to be trained and certified in software security. The client or customer should be aware of the need and importance of incorporating security into the software product they request. The requirements analyst should be trained to solicit and translate functional requirements into security requirements. The project manager should be versed in making necessary project-related decisions appropriate to security within the constraints of scope, schedule, and budget. The coder should be trained to write secure code and the tester should be trained to validate that the code is secure. Operations personnel should be trained in least privilege computing and skilled in monitoring and disposing of software securely. Effective training and education should target changing the behavior of these influencers to include security in the software by default.

Conclusion

With software deeply impacting our everyday lives, the need for it to be secure is an absolute necessity. The real cost of insecure software is not merely the quantifiable fines imposed on the negligent, but also the loss in customer confidence, the loss in reputational damage and brand, loss that, in many cases, is irreparable. Playing the blame game as to who is truly responsible for insecure software is reactive and not as effective as building software securely by weaving security processes through the software lifecycle. From requirements gathering to disposal, security should be built into the software.

A new culture reflecting a change in the mindset of those involved in the SDLC is necessary. This culture should promote security in the SDLC while understanding the risk of software built without security in mind. Awareness, education, and certification programs built around security in the SDLC are critically necessary, and (ISC)^2's Certified Secure Software Lifecycle Professional (CSSLP^{CM}) certification program may be the harbinger in creating this culture and addressing the need for secure software. Such a change in people's mind will be the first step in *Being Secure in an Insecure World.*

About (ISC)²®

The International Information Systems Security Certification Consortium, Inc. [(ISC)²®] is the globally recognized Gold Standard for certifying information security professionals. Founded in 1989, (ISC)² has now certified over 60,000 information security professionals in more than 130 countries. Based in Palm Harbor, Florida, USA, with offices in Washington, D.C., London, Hong Kong and Tokyo, (ISC)² issues the Certified Information Systems Security Professional (CISSP®) and related concentrations, Certified Secure Software Lifecycle Professional (CSSLP^CM^), Certification and Accreditation Professional (CAP®), and Systems Security Certified Practitioner (SSCP®) credentials to those meeting necessary competency requirements. (ISC)² CISSP and related concentrations, CAP, and the SSCP certifications are among the first information technology credentials to meet the stringent requirements of ANSI/ISO/IEC Standard 17024, a global benchmark for assessing and certifying personnel. (ISC)² also offers a continuing professional education program, a portfolio of education products and services based upon (ISC)²'s CBK®, a compendium of information security topics, and is responsible for the (ISC)² Global Information Security Workforce Study. More information is available at **www.isc2.org.**

"Security is just another attribute of your software like usability, performance, reliability, scalability, etc. The idea of incorporating security into the SDLC begins with evaluating the relative importance of this attribute on an application and then going on to evaluating and incorporating controls in-line with that."

Talhah Mir
Sr. Program Manager Lead
Information Security Awareness

About the Author

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA is CEO and President of Express Certifications and SecuRisk Solutions, companies specializing in professional training, certification, security products and security consulting. His security experience includes designing and developing software security programs from Compliance-to-Coding, application security risk management, security strategy and management, and conducting security awareness sessions, training, and other educational activities. He is currently authoring the Official (ISC)² Guide to the CSSLP, is a contributing author for the Information Security Management Handbook, writes periodically for Certification, Software Development and Security magazines and has contributed to several security topics for the Microsoft Solutions Developer Network. He has been featured in various domestic and international security conferences and is an invited speaker and panelist in the CSI (Computer Security Institute), Catalyst (Burton Group), TRISC (Texas Regional Infrastructure Security Conference), SC World Congress, and the OWASP (Open Web Application Security Project) application security conferences. He can be reached at **mano.paul@expresscertifications.com** or **mano.paul@securisksolutions.com.**

a A Tax on Buggy Software.
http://www.forbes.com/2008/06/26/rice-cyber-security-tech-security-cx_ag_0626rice_print.html

b Application Security: Everybody's Problem.
http://www.burtongroup.com/Research/PublicDocument.aspx?cid=763

c Best Practices In Data Classification For Information Lifecycle Management.
http://www.sun.com/storagetek/white-papers/Best_Practices_Data_Classification_ILM.pdf

d Risk Triage and Prototyping in Information Security Engagements.
http://www.cisco.com/web/about/security/intelligence/risk-triage-whitepaper.html

e NIST 800-64 REV 1. Security Considerations in the Information Systems Development Life Cycle.
http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf