



Duct Tape, Band-Aids and Bubble Gum Shouldn't Be Used to Build Security

A WHITE PAPER PROVIDED TO ASPE BY SECURITY INNOVATION

Duct Tape, Band-Aids and Bubble Gum Shouldn't Be Used to Build Security

Joe Basirico
Security Analyst

Security Innovation

 **ASPE**
SDLC TRAINING **Presents...**



PREPARED BY
SECURITY INNOVATION[®]
THE APPLICATION SECURITY COMPANY

Security Innovation, Inc.
187 Ballardvale Street, Suite A170
Wilmington, MA 01887
+1.978.694.1008
www.securityinnovation.com

Throughout my time spent delivering talks at security conferences and in classrooms of prominent software companies, I consistently hear the same question: “What can we do to secure our software completely?” This question typically comes in response to feeling completely overwhelmed by all the best practices, methodologies, and techniques to which my audience has just been exposed.

SDLC by Any Other Name....

Unfortunately there is no single or easy answer to this question. Security must be dealt with and considered in every phase of the software development lifecycle. I know practitioners would rather that I discuss the silver bullet solution, but there is none. Security in the SDLC is essential – so much so that Microsoft has bestowed upon it a completely new name: the Secure Development Lifecycle, or SDL. While these acronyms may be similar, the way we think about software differs drastically in each model. The original software development lifecycle was created to deliver applications that are feature-rich, free of bugs and are shipped on time. It was designed at a time prior to security being the prevalent issue that it is today. The Secure Development Lifecycle attempts to marry the pillars of the original SDLC with fundamental secure practices throughout the lifecycle. This practice will create an application that has a secure core and can better withstand attacks.

When software development teams begin to think about security early in the SDLC, it reforms our ideas of what our software will look like and gives us flexibility to address the concerns of not only security, but performance, usability, reliability, maintainability and many other important metrics. It's only when we attempt to tack on security three weeks before the ship date that it becomes impossible. There are five indispensable things to consider when designing software:

1. Security principles must be applied to every phase of the SDLC, otherwise ship dates will slip and software will remain insecure.
2. People not professionally trained in security cannot be expected to make sound security decisions.
3. Software will never be 100% secure and people will inevitably make mistakes when implementing a secure system. Therefore, proper fail-safes must be in place to mitigate these risks.
4. The insider threat is real; I've worked in both small and large companies and my distrust for those protecting data is the same.

The deck is stacked against the software development team. Attackers only have to find a single vulnerability; you have to find them all. Additionally, auditors and compliance are here to stay – and they want proof that you are taking the necessary steps to produce a secure application.

Requirements Gathering Phase

In the first phase of any project the Project or Program Manager (PM) must request feedback from his/her users to understand what new features to include in the next version. Often this phase requires countless hours of creating customer surveys and use-case scenario planning. Even before the application is conceived, security must be taken into account. The PM can help facilitate this by asking clients such leading questions as:

- ✚ How bad would it be if this piece of personal data was lost?
- ✚ How tolerant are you to password policies?

- 🚩 Would you be interested in using a biometric scanner, if it meant an increased level of security and confidence?

Asking these questions will do two things for you. First it will instill confidence that you take security seriously and want to protect customers' sensitive data. It will also ensure that as features are created in the next phases you will know how tolerant your users are to security mechanisms or data loss. Armed with this intelligence, you can now begin writing requirements.

Requirements Authoring Phase

Like a sculptor sketching an outline of a new sculpture, the PM must begin thinking about how each component will fit together in this phase. If security is designed into this phase the PM can consider not only what should happen during normal execution, but what should not happen and what the consequences are should a bad thing occur (i.e. a data breach). Before the first line of code is written and the first process is run, the PM needs to think about possible attacks, vulnerabilities and mitigations. If the sculptor Alexandros of Antioch had anticipated children swinging from her arms before sculpting her, the Venus De Milo might still be completely intact today.

Effective exercises to create security requirements are:

Consider the attacker in each feature – Consider not only what a benign user will create with your software but also what an attacker may be able to do with it.

Outline what each feature should do and what it should not do – One requirement for a login control may be to ensure that a user with valid credentials is allowed access to his/her data. Another requirement for that feature should be that the login control will disallow any unauthorized user from retrieving another user's personal data.

Understand the principle of "secure by default" – When designing the system, ensure features are off until a user enables them. This will decrease the application's attack surface. A disabled feature does not provide an avenue of attack, and also has the added benefit of increasing performance and lowering the memory footprint.

Use Threat Modeling to understand the possible threats to a system – Threat modeling and threat trees are great ways to understand the security implications of each feature. Threat modeling is an effective risk analysis technique that will be described near the end of this article.

Design Phase

Architects translate the features wish list into a structured document that developers can use to create an application. This document explains how the pieces of the application will fit together. The architect will consider the best technologies to use and how each component will communicate with each other. In this phase the architect must consider the attacks, common vulnerabilities and best practices mitigations of each component individually, as well as the security of each communication channel. The architect must be all-knowing. Key activities to consider include:

Review and update the threat model document – The threat model should be a living document, which contains all the assets, entry points and attack vectors of the software. Ensure there are mitigations in place for each attack in the threat model.

Ensure communication channels are secured – Use proper encryption standards such as SSL and TLS for web traffic or AES for transmitting blocks of sensitive data.

Enable the developer to understand how the component he/she will be working on will be used in the application – Consider future implementations to be sure that if a component is used in a different way than originally thought, it does not expose a latent security flaw. Ensure that each function or feature is responsible for checking all inputs or flag the function to be only used behind a secure trust layer.

Development Phase

This phase is often a black hole, where the vast majority of vulnerabilities creep in. Developers sometimes believe that the operating system or the libraries they call are perfect and cannot fail- but we know this is not the case.

There is a lot to consider when developing an application, so developers must be trained in secure development and must show a constant focus on security. This includes a careful examination of the libraries and functions they use. The moment a developer forgets to check a return value or fails to validate input, he/she will likely be exposing serious security vulnerability.

To make things easier and to increase the overall quality of software here are some best practices:

Train your stakeholders – The most important thing developers can do is to become knowledgeable about the system they are securing. Without the right knowledge they cannot know the security implications of their decisions. The end of this article lists helpful books and training courses to learn more.

Review the threat model – Leveraging the threat model as a guide for development is a great way to make sure each threat and possible vulnerability has been mitigated. The Threat Model should always be updated as new threats are discovered or mitigated.

Review code for security – Code reviewing can be difficult to get traction on because it requires so much time. However, just like an author wouldn't dream of publishing a book before an editor had a chance to review it, a developer should always have a second pair of watchful eyes examine sensitive code blocks.

Use static analysis tools – Static analysis tools can help create more secure code by quickly and easily checking for common security vulnerabilities. However, these tools are in their infancy and you need to understand what they can and cannot do.

Use tried-and-true security libraries – Security libraries are very difficult to write. Leveraging the libraries that have been provided for you will reduce development time and cost while increasing the security of your application. Attempting to create a new encryption or hashing algorithm is not a good idea. There are proven algorithms and libraries that have been peer-reviewed and scrutinized for possible weaknesses. At the time of this writing the standard for encryption is the Advanced Encryption Standard (AES 128, 192, and 256). The standard for hashing is SHA-256 or greater. All other algorithms have shown signs of serious weakness and should be discontinued.

Do not implement your own security mechanisms – As mentioned above, leveraging existing security mechanisms for cryptography, authentication and integrity will help reduce costs and development time while creating a more secure system. No security will be gained from creating a proprietary and otherwise unknown, security algorithm.

Validate your input – Input is the root of all evil. Without input our software would do exactly the same thing every time, which would be less glamorous, but predictable and safer. Consider all methods of input including: the UI, registry keys, the network, Remote Procedure Calls, XML, HTTP requests,

UDP, return values from APIs, corrupted memory, files, and every other way your application knows what to do.

Test Phase

the tester is the guardian of the application. The role of the guardian is not easy. Compressed ship dates and slippages in the other phases can make the life of a tester difficult. We often hear "It's OK, we'll make this time up in testing." But it's impossible to fit security testing into an already abbreviated test phase.

There are three main characteristics of an adept security tester:

1. **Imagination** – A tester must be able to think about software and software security from a completely different perspective, including the ways that the software was intended to be used and considering all other ways that the software could be used. Think like my mother, think like an attacker, think like a power user. Use your imagination to explore every corner of the software to find vulnerabilities that are elusive and likely to wreak havoc if discovered by a malicious user.
2. **Knowledge of system** – An effective security tester understands how every component fits together and can recognize when things are out of place. A complete knowledge of the system will alert the tester when components are used improperly or when security vulnerabilities manifest themselves in difficult to find ways.
3. **Evil streak** – You cannot discount the benefit of being able to think like an attacker. It's important to not consider the obvious scenarios such as "Are there any temporary files being created where they shouldn't be?" Think more destructive and evil: "What could I do with these temporary files if I found one?" The opportunity to be evil without doing anything wrong is what makes security testing so exciting.

Below are a few points to help get the imagination, knowledge and evil streak going.

Education – The single most important aspect of sound testing is a solid foundation in security. This can be provided by books, classes, conferences. The key, though, is never falling into the trap of thinking you know enough.

Books are often written by experts in the field with lots of hands-on experience (and you can always refer to them when needed).

Security classes allow you to get real hands-on testing tips and knowledge from security professionals. The techniques that you learn will stick with you and become part of your daily testing routine.

Conferences are great opportunities to communicate and knowledge share with others in the security space

Review the threat model – The threat model can be used to hold the PM, architect and developer's feet to the fire. It's a great place to start, but great testers know that real vulnerabilities reside outside of this model. Be sure to use your imagination to isolate new threats and vulnerabilities, and be sure to update the Threat Model as they are discovered.

Test with a methodology – Banging your head on a keyboard has never provided a reliable or repeatable method for finding vulnerabilities that the end user cares about or the attacker may exploit.

For this reason employ a methodology to look for vulnerabilities in locations where they are likely to be found and where they are likely to cause the most damage. A methodology, however, is not a test plan; don't let it stifle your nose for great bugs. Testers are like the police dogs of the software world, if you sniff out a great SQL injection vulnerability, feel free to explore it.

Release and Documentation

Once testers bless the application, it is time to release the product to customers. Similar to sending a child off to college, this can be a heart-wrenching, but relieving experience. Support and documentation for customers must be provided so they can deploy software securely and seek answers to problems they encounter. In this phase, disabling every security feature while enabling every functional feature creates a highly functional product, but increases the attack surface of the application exponentially.

Before releasing a product be sure to follow these steps to make it easy for your customers to deploy safely:

Document all security features – Explain the benefit of leaving these features on and how to allow legacy features to communicate.

Document ways to turn on features that are not available by default – Provide easy to follow instructions on how to turn on disabled features securely.

Train support staff – Be sure to train the support staff on the new security features and the best ways to use them. Make sure the support staff does not adopt a “turn everything on and walk away” policy.

Bringing it all Together

The best organizations unite all of these techniques and understand that assessing their applications before they ship is a prerequisite. Doing this provides a clear understanding of the risks involved with shipping and its impact to the end user. Deploying mitigation techniques before shipping ensures that risks found in their assessment have been minimized. Proactive organizations also ensure that every member of the team has the proper education and security charter that enables the development team to effectively incorporate security at each phase of the SDLC.

Knowledgeable development teams can make informed decisions when presented with day-to-day problems. Having each member of the team trained in security is important and will help ensure that your application is significantly more secure, but this is not sufficient. While the PMs, architects, developers and testers strive to create a secure product, they should also employ key security practitioners at each phase dedicated to reviewing their phase's deliverables. These individuals ensure rubber hits road and maintain a steady, repeatable secure development process. Typically, a developer spends the majority of his/her time reviewing code for security vulnerabilities; a tester focuses on testing for unmitigated threats; and an architect ensures that the application as a whole can be securely produced.

Finally, proactive organizations understand the benefit of partnering with companies with specific expertise in assessing security risk, mitigating threats and providing the knowledge their teams need to discover vulnerabilities and understand the risks they pose.

With this multilayered approach, software ships more securely, closer to the scheduled delivery date and closer to anticipated cost. Your company becomes more competitive in the market and reduces its overall risk, resulting in fewer headaches from the auditors and customers.

More Information for Project Managers

Threat Modeling by Frank Spiders and Window Snyder

More Information for Developers

Writing Secure Code 2nd Edition by Michael Howard and David LeBlanc

Secure Programming Cookbook for C and C++ by John Viega and Matt Messier

Creating Secure Code training course by Security Innovation

More Information for Testers

How to Break Software Security by James Whittaker and Herbert Thompson

Hacking the Art of Exploitation by Jon Erickson

How to Break Software Security training course by Security Innovation

How to Break Web Software training course by Security Innovation

RSA security conference <http://www.rsaconference.com/>

Software Security Summit conference <http://www.s3con.com>

About the Author

Joe has spent most the majority of his professional career studying security and developing tools that assist in the discovery of security vulnerabilities and general application problems. His primary responsibility at Security Innovation is to deliver security courses to Software Teams in need of application security expertise. He has trained developers and testers from numerous world-class organizations such as Microsoft, HP, EMC, Symantec, Liberty Mutual, Sony, State Farm, Credit Suisse, Amazon.com, Adobe and ING.

Joe is also responsible for participating in customer security process assessments as well as security engineering activities such as security design reviews, security code reviews, and security testing and security deployment reviews.

Joe has been interviewed on several occasions by media outlets including *SC Magazine* and *Software Test & Performance*. He maintains a bog with *CSO Magazine* and has written several publications that focus on vulnerabilities at the source code level, including:

- ▶ "Application security shouldn't involve duct tape, Band-Aids or bubble gum" – *SearchSoftwareQuality.com*
- ▶ "Scan your Source Code to Locate Weak Spots Early - *Software Test & Performance*
- ▶ "Data Mining Precautions for Web App Security: Or, How I Learned to Stop Worrying and Love the Data" *DM Review*
- ▶ "White box Security Testing Using Code Scanning" - *Dr. Dobbs Journal*
- ▶ "The Perils and Possibilities of Patching" - *CIO Update*
- ▶ "The Dangers of Data Mining" - *DM Review*
- ▶ "Anatomy of an Attack", "Hacker Report: Static Analysis Tools, and 'Hacker Report: Vulnerability Scanners" – Security Innovation Intelligence Reports

Joe is a seasoned practitioner and researcher in the field of incorporating security into the SDLC and a highly demanded presenter and for the topic of software development best practices. He has delivered presentations at several world-class venues, including:

- ▶ *The Five Most Dangerous Application Security Vulnerabilities - and How to Test for Them.* Software Test & Performance, November 2006.
- ▶ *How to Break Software.* NationWide Insurance Testing Symposium Keynote, September 2006.
- ▶ *Building Secure & Reliable Web Applications.* Software Security Summit, June 2006.
- ▶ *The Five Most Dangerous Application Security Vulnerabilities - and How to Test for Them.* SecureWorld Seattle, September 2005.
- ▶ *Building Secure Applications in an Insecure World.* OJ.X 2005, Compuware's annual application development conference, October 2005.
- ▶ *Holodeck – an Introduction to Fault-Injection.* Microsoft Professional Developers Conference, October 2005.

Joe holds a B.S in Computer Science Graduate from Montana State University.

About Security Innovation

Security Innovation is the authority on application security and a leading provider of risk analysis, risk mitigation and education services to mid-size and Fortune 500 companies. Global technology vendors and enterprise IT organizations rely on our services to identify security risks in their software systems and development processes and facilitate the changes needed to mitigate them. The company is headquartered in Wilmington, MA and has offices in Seattle, WA and Amsterdam, the Netherlands.