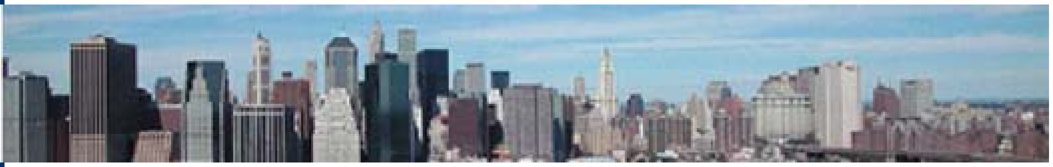

89 Fifth Avenue, 7th Floor
New York, NY 10003
www.TheEdison.com
212.367.7400

White Paper

Microsoft Security Development Lifecycle Adoption: Why and How

September 2013

First Publication: September 2013
Produced by: Glenn Graham, Senior Analyst; Barry Cohen, Editor-in-Chief; Manny Frishberg, Editor

Table of Contents

Executive Summary

This white paper reviews how members of the financial services industry are using the Microsoft SDL (Security Development Lifecycle). Specifically it shows how members of BITS, the Technology Policy Division of the Financial Services Roundtable, have learned and benefited from the Microsoft SDL and are using the BITS Software Assurance Framework, both of which help to meet or exceed the guidance in ISO/IEC 27034-1 Standard.

This paper was developed following in-depth interviews with participants representing some of the leading banks and financial services companies in the United States. Their chosen architectures included both Microsoft-centric and open source. The adoption maturity ranged from highly refined through years of implementation, to a brand new adopter about to begin integrating the SDL into their organization. While we don't show explicit quotations, all the information in the case studies is directly from the interview content.

The quantitative and qualitative benefits of using the Microsoft SDL range from reduced development costs and time, to more secure applications that leverage the critical training and knowledge across the entire development organization. No longer is it necessary to teach each developer the entire realm of knowledge to be able to manually develop secure applications. Also, it reduces the need to depend on each programmer to purposely program using secure coding practices. Instead, the infrastructure and the overall development system enforces secure coding through the development methodology and confirms it through both manual and automated testing. This assures corporate management and customers that the applications are secure, and also that new threats are dealt with and resolved quickly.

Not only is the SDL integrated into a company's internal development lifecycle, but it should also be applied to all software applications that are used by the company, no matter the development source. The SDL is a framework for the entire development process. Companies should require the use of an SDL which meets or exceeds the guidance in ISO/IEC 27034-1. This will help to ensure that all software used by the organization is developed securely, and provides a common language for discussing secure development practices.

By using the SDL's processes, having a team approach to developers' skill and knowledge integration, and an ongoing training plan, an organization can achieve a level of application security that would not otherwise be possible with the same resources. The SDL is all about process and framework that must be incorporated into

your overall Software Development Life Cycle (SDLC). The SDL is a "force multiplier" in that it increases the benefits of adopting best practices. However, the SDL is not an objective in itself, but an ongoing process of continuous achievement.

The BITS organization addresses issues at the intersection of financial services, technology and public policy, where industry cooperation serves the public good, such as critical infrastructure protection, fraud prevention, and the safety of financial services. BITS is the technology policy division of The Financial Services Roundtable, which represents 100 of the largest integrated financial services companies providing banking, insurance, and investment products and services to the American consumer.

If you are interested in joining the BITS organization of the Financial Services Roundtable to facilitate with many companies who have also implemented the Microsoft SDL and the Software Assurance Framework, please find more information at http://www.bits.org/about/membership.php.

Introduction

Objective

The goal of this white paper is to show the usage of Microsoft Security Development Lifecycle (SDL) and how it has been integrated into the Software Design Life Cycle (SDLC) of financial services companies. It describes the business benefits of using the Microsoft SDL, along with adoption approaches and integration methods. Without divulging competitive methods and sensitive information, we describe observations, objectives and approaches used to implement SDL. We have provided two case studies– one incorporating the Microsoft SDL into a Microsoft development environment, the other applying the SDL into an open source development environment.

Since the SDL is a process for the entire development organization, this paper would be an excellent introduction for all functions, from the executive sponsors to the entire development organization. Many of the supporting documents mentioned in this paper are dozens of pages long and highly technical. We have tried to make this paper as understandable to the non-technical businessperson as to the developer.

Due to the need to keep interviewees confidential, a general description of the generic types of financial institutions who participated in this survey is provided below. But this description should allow adoption candidates to see that organizations similar to their own in the financial services sector have found value from implementing Microsoft's Security Development Lifecycle.

The survey participants in the interviews for this paper represent some of the leading banks and financial services companies in the country. Their chosen architectures included both Microsoft-centric and open source. And the adoption maturity ranged from highly refined through years of implementation, to a brand new adopter about to begin integrating the SDL into their organization. While we don't show explicit quotations, all the information in the case studies is directly from the interview content.

In several cases, the leaders of the SDL implementation had multiple successes with more than one company. And while they were repeat implementers, they still began each company's implementation process with an educational refresher for themselves.

Audience

This report is for IT and development executives, and IT security officers at financial services organizations, who make acquisition and development decisions, whether development is in-house or outsourced, or applications, are purchased and customized.

This paper is intended to be of value whether you are a veteran adopter of the SDL or a first time implementer. Using the SDL can confirm your accomplishments and help you to move toward greater benefits. For someone just beginning the process, this paper provides a road map to getting started, drawing on the experiences of those who have already traveled the same path.

As an additional valuable reference, the BITS Software Assurance paper may be obtained at the following URL:
http://www.bits.org/publications/security/BITSSoftwareAssurance0112.pdf.

Why use SDL in Financial Services Software Development?

The Security Development Lifecycle (SDL) is a security assurance process that is focused on software development. Combining a holistic and practical approach, the SDL introduces security and privacy throughout all phases of the development process to reduce the number and severity of software vulnerabilities.

In the past, application development, and more specifically the coding associated with software development, was a somewhat separate function; functional requirements were presented to developers and a finished application was then turned out that satisfied those business requirements. Today, it is much more complex and sophisticated for a myriad of reasons. This increased complexity makes the need for a practical approach to developing software in a secure manner even more necessary.

Application Security

Security in the financial services industry has obviously always been a great concern. The industry deals with money and people's personal and financial information. But, as has become evident by the constantly changing ways of hackers, security risks are pervasive throughout all software applications. In the beginning, malicious and mischievous hackers would attack the primary financial institutions directly through their core applications. Now, more and more hackers are getting in through the individual user, often entering via non-related applications on an individual's PC, smart phone, PDA, Web-enabled TV, or other device. Once inside, they can masquerade as that individual to enter the secure application through that user's secure connection. This creates many new challenges for developers of financial services applications, who now have to continuously detect illicit activity, not just have a robust initial entry authentication. Now, every module and every major function needs continuous authentication and authorization.

Training costs, development efforts, testing, and support to get all developers to manually inject this type of secure programming element into every bit of code they write are prohibitive. A more effective approach has to be utilized. Over the last few years, industry leaders in the financial services sector implementing security in their applications have interwoven secure programming and development practices and solutions throughout their Software Development Life Cycle (SDLC). A major element of that is incorporating Microsoft's SDL into their SDLC.

First, let's examine some of the challenges that had to be solved to reasonably incorporate a secure development methodology into the overall SDLC. Increasingly, more sophisticated threats and the longtime standby threats are becoming more robust as the Internet and computing horsepower increase. DDoS (Distributed Denial of Service) attacks are still the most prevalent type, and although they can be mitigated to help prevent permanent harm to data or any loss of confidential information, they still block access to critical business systems and prevent customer interactions.

Standards Compliance

The SDL helps to support an organization's conformance with standards such as **ISO/IEC 27034-1**, which defines application security not as the state of security of an application system but as "a process an organization can perform for applying controls and measurements to its applications in order the manage the risk of using them." Appendix B contains an explanation of **ISO/IEC 27034-1,** including the section **ISO/IEC 27034-1:2011,** and information on where it can be obtained.

There are many different standards that an organization has to be in compliance with, depending on their industry, and products or services. The Microsoft SDL contains processes and steps which could help an organization achieve many of these standards.

Although many organizations sometimes like to try and do things internally, with no outside help, there is no reason to approach SDL in that way. Many organizations have already struggled through everything that you will face when implementing SDL for the first time. In some cases, they have completely revamped their SDLC and can share all their lessons learned with others. So, take advantage of what Microsoft, the BITS organization, and individual adopters can provide as you move forward with a successful implementation of SDL.

How to Implement SDL

Education

The first step in implementing SDL is education. That means learning what SDL is, how it is utilized, and how it can be integrated into an organization's SDLC. Those who are now using SDL say that education was initially the key to doing anything constructive with SDL. Whilst the SDL can frequently be overlaid on an organization's existing SDLC without any re-engineering required, in some cases, significant "clean-up" to the organization's overall SDLC had to occur because, while they may have had a defined SDLC, it might have been outdated or not followed to the extent that enhancements to it were directly reflected in the daily activities of the developers.

The "best of breed" implementers of the SDL first went to Microsoft directly (either by utilizing the resource on the SDL website, or by engaging Microsoft Consulting Services) and immersed the leader of their endeavor in the latest information about the SDL—for the most accurate information, go to the source. Being a very large and experienced software development organization, Microsoft is uniquely qualified to share this type of knowledge. In most cases, improving some aspects of the internal SDLC was also necessary to fully embrace the SDL. That means that the rigors of implementing the SDL resulted in an overall improvement in the SDLC.

Customization

OK, let's say it again. The Microsoft SDL is a ***process*** to apply to the Software Development Life Cycle (SDLC). And this means taking the parts of the SDL that apply to your particular development environment and products.

If you are not experienced at modifying your SDLC, or maybe do not really have one, then you should seek out qualified professionals to assist you. Not having an appropriate SDLC can be a recipe for disaster when trying to utilize the SDL. In fact an insufficient SDLC causes problems in general, with or without incorporating the SDL into your environment.

Tools and Infrastructure

As well as a suitable development environment, the project team can take advantage of tools and guidance available on the Internet. Microsoft offers a variety of free-of-charge tools and guidance to complement the various phases of the development lifecycle on the SDL website at www.microsoft.com/security/sdl/adopt/tools.aspx.

Training

Once you have decided how to integrate the SDL throughout your SDLC, you should provide some training for the various roles and responsibilities in your organization. While a general introduction session may be fine to familiarize everyone, specialized classes for different functions should highlight the elements that apply to their particular job functions or programming responsibilities.

Do not try to train your staff on every aspect of the SDL before having them begin to use it. Whether by examples or real-life projects, people have to actually use it as the first step toward it becoming a way of life.

Do It!

There is simply no substitute for doing it. All the training in the world never resulted in a single line of code. Real development must occur using the newly defined SDLC incorporating the SDL.

This also will show which aspects may have to be refined better for the needs of your organization. If the developers are struggling to embrace the tenants of the SDL, then you may have made it too complex, or improvements to the SDLC may be needed.

Measurements

How do you know if a process is working? Results must be measured. The SDLC needs measurement procedures that can detect security issues in a given application, and correlate between security issues in your application and the developers who wrote the code that caused the problem. The SDL includes two activities which help to measure the effectiveness of the process. The Final Security Review is performed just before the software is released, and usually includes examining threat models, tools outputs, and performance against the quality gates and bug bars defined during the project. After release, a project post mortem is conducted to gather any insights or learning that were uncovered during the project and this analysis is used to improve the process for the next project.

Each company may have to come up with its own customized approach to determine the developers' productivity and quality, as there are different ways to look at these metrics.

Review

Finally, once you have done all this and have measurement data compiled, you should conduct formalized review sessions. By formalizing these sessions and using the results to encourage self-improvement you are bringing the involved parties into a constructive feedback loop.

Enhancing a Successful SDL Integration

Even after an organization has embraced SDL and integrated it into their SDLC, there are still many things to do on an ongoing basis for additional benefit.

Microsoft has an SDL Optimization Model. In one dimension, it has the sequential step flow of the SDL; in the opposite dimension, going across the flow, is a maturity model of Basic, Standardized, Advanced and Dynamic levels. This illustrates that even after full implementation of the SDL, there is room for significant improvement of how it is being used. Some of that improvement requires feedback from existing projects. So, fully implementing the SDL on the initial flow is impossible. Reviewing the results and feedback of your SDL implementation obviously comes after you have completed at least one project.

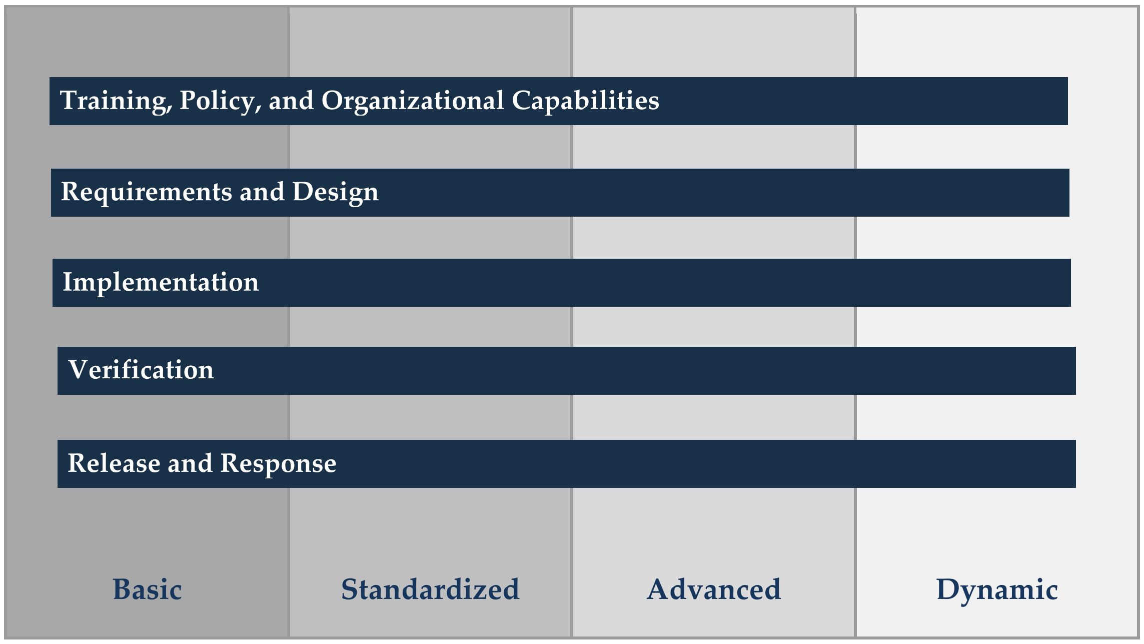

Figure 1: SDL Optimization Model with capability and maturity levels

In contrast to other software maturity models, the Microsoft SDL Optimization Model focuses strictly on development process security improvements. It provides prescriptive, actionable guidance on how to move from lower levels of process maturity to higher levels, and avoids the “list of lists” approach of other optimization models.

Case Study #1:
SDL in a Microsoft Development Environment

This first case study of implementing the SDL is in a Microsoft development environment, where the primary Integrated Development Environment (IDE) is Visual Studio and the target architectures are ASP.net, Java and C++ applications, primarily using a SQL Server database. The application portfolio is comprised of a mix of custom developed and modified packaged software. The experiences and scenario presented are a compilation from multiple interviewees, but focuses on the most experienced implementers.

Having done multiple SDL adoptions, and with years of software development experience, they have achieved a more refined and successful result in their most recent implementation. Each line of business has its own SDLC, with SDL integrated into it. The objective is not to force all lines of business to have an identical SDLC, but rather to fully integrate SDL into their SDLC.

Not all developers are programming with the latest languages and environments; COBOL, FORTRAN and other legacy applications are still being maintained. Also, new developers are constantly joining the organization, and usually have not been trained in an SDL-integrated SDLC. Therefore, there is a vigorous ongoing training program.

The results of testing are mapped to each individual developer. This enables training each developer so they understand how their coding was deficient and caused security flaws. If the developer continues to make the same errors time and again, after being informed of their defects, and how to fix the problem, then that developer may not be sufficiently capable to remain a productive developer.

Multiple training curriculums were developed, with both common lessons for everyone and specialized lessons that are focused on a particular discipline's needs. Compliance with training requirements is strictly enforced and monitored to ensure that each developer has the knowledge and skills they need to do their job. Cross training provides redundant knowledge and enhances developer skill. Some, who have a special aptitude for security, are designated as Security Champions and provide expert support to others within their team.

Once the implementation was completed, multiple software sector experts, including some from Microsoft, were called in to critically evaluate their result and provide feedback. This self-review helped the continuous improvement process.

Case Study #2:
SDL in an Open Source Development Environment

The Microsoft SDL is as completely applicable to an open source development environment as it is to a Microsoft architecture environment, because it is a framework that guides the development process.

While most of the organizations who were interviewed have some open source applications, many are Microsoft shops and only use open source, including the LAMP stack, for non-critical applications and static web sites. But even for those, any application that is part of their overall world is a potential entry point for hackers. So, while not as critical and a lower priority threat, they still must be considered as applications that should have the SDL applied to them.

The development environment in an open source organization is usually Eclipse, instead of Visual Studio. Static analysis is part of the IDE and automated testing tools are used. The open source world has a number of choices for testing tools, but similar to the Microsoft environment, it can take three to four days to get the results back from a test instance. So, daily builds are not able to be tested and have the results back to the developer before the next build occurs.

The goal of this SDL implementation is to have minimum overhead of the security-related testing tools, so that the developers can focus on writing code. The desire is to train the developers on how to use the tools, rather than having to become security experts. They can let the integrated tools find the flaws in the code and highlight them in real time, as the developer is writing code.

As in the first example, education and training is the biggest area of resource expenditures. But, in the open source world, there is often a greater choice of tools for testing and similar tasks. Therefore, a smorgasbord of tools and techniques was compiled that are used based on risk prioritization.

Whether open source or proprietary software such as a Microsoft architecture, the SDL can be applied the same way and with equal benefit. Testing tools are different to integrate into different development IDEs. But there is no difference in terms of the threats that are being thwarted and the coding weaknesses.

Adopters Best Practices

Here are some of the best practices of those who have adopted the Microsoft SDL:

- Empower all your people! First, sell the philosophy to everyone involved.
- Educate all functions that have any interaction with the development process, from analysts to DBAs, with customized curriculums that focus on their functional needs.
- Train and educate *all* developers, both internal and external. Developers learn by doing: the more they do, the better and faster they learn.
- Apply SDL to the early stages, so that SDL is applied to the design, development and testing processes.
- Use automated testing tools as much as possible, in conjunction with manual testing.
- Streamline test points to enable completion of testing between builds.
- Track faults to each developer, so remedial training can be provided and eliminate recurrence. Create a system to manage and track the correlation between developers and coding faults that the testing tools discover.
- You must do full coverage testing. Otherwise, there may be weak areas of code through which threats can enter your entire world of applications. If you need to prioritize testing, focus on the threat model and attack surface.
- Establish development teams, with a pyramid of highly trained and experienced developers at the top, who can help others by being the expert reference.
- Trust that the SDL does work. Be proactive and implement it fully.

Lessons Learned

Here are some of the key lessons learned by adopters of the Microsoft SDL:

- Principles and paradigms are universal. It’s the implementation that is different.
- The only way developers really learn, and that you can ensure that the developers are using all aspects of the SDL framework, is to have them work with it hands-on, as soon as possible. That includes lots of hands-on work during training.
- If an Agile process with daily builds is used, static scanning across all the developers’ work may not keep up, as it can take more than a day to scan a build and report. So, integrated scan tools that are a part of the developer’s IDE are needed to keep up with the pace of development and daily builds. Sprint cycles of one week or greater can be handled by today’s tools.
- Using SDL has enabled a development organization to educate and train their developers to become proactive instead of reactive, when it comes to defensive programming against security threats.
- The benefits of incorporating SDL into the SDLC are immediate because developers begin to code with fewer security vulnerabilities, which reduces rework required after code review and testing.
- Designating SDL champions within development groups causes the SDL be more readily implemented and the developers to become internally motivated, which ensures the active adoption of the SDL into the ongoing development activities.
- The SDL is a framework that must be integrated into your SDLC. So the cost to implement the SDL is primarily for training, and implementation support for the methodology and testing tools.

How to Get Started with SDL Integration

OK, now you have an idea of what Microsoft's SDL is, how to implement it in general, and many other things. But there are still some things that are confusing, and even the basic first step may still be unclear.

First, go to Microsoft's web site and the home page of their SDL portal at http://www.microsoft.com/security/sdl/default.aspx to read about it.

The Microsoft portal has all the information available on their SDL. You could start by reading about the simplified implementation of the Microsoft SDL. You can download free tools and templates to jumpstart your SDL implementation, leveraging the training, consulting, and tools expertise of Microsoft Services and the SDL Pro Network, if needed.

Probably the most useful web page is: "Simplified Implementation of the SDL:" (http://www.microsoft.com/en-us/download/details.aspx?id=12379) with downloadable versions, as both Word and Excel files.

This document illustrates the core concepts of the Microsoft Security Development Lifecycle (SDL) and discusses the individual security activities that should be performed in following the SDL process. The Simplified SDL guidance is also available under an Excel spreadsheet format, and lists the 16 mandatory SDL security practices, along with implementation details and resources for each practice.

This "Simplified Implementation of the SDL" paper presents:

- A brief overview of the Microsoft SDL.
- An overview of the Microsoft SDL Optimization Model with particular attention to where the Microsoft SDL fits within the Optimization Model.
- A discussion of individual Microsoft security development practices, including: roles and responsibilities for individuals involved in the application development process, mandatory security activities, optional security activities, the application security verification process.

Conclusion

After conducting detailed interviews with SDL adopters, ranging from veterans of multiple implementations to a first-time user just beginning the planning stage, Edison Group has arrived at several conclusions.

- First and foremost, the SDL clearly is beneficial for any software development organization to adopt, especially for anyone whose business involves applications with great security risks, such as financial services.
- BITS is the "must join" organization for any entity who develops and uses software applications in the financial services sector, as its membership is comprised of 100 of the largest integrated financial services companies.
- Training and education about the SDL process is essential to implementation, as it is a quite complex process that must be incorporated into existing SDLCs.
- While the SDL shows a traditional, sequential SDLC, it can be applied to any and all software development methodologies. In fact, it is even more essential for the various rapid application development methodologies.
- The SDL is agnostic to the chosen software architecture. It can be applied to any vendor architecture and framework, including open source tools and solutions.
- Users have Agile sprint cycles, ranging from daily builds to a build every two weeks. If not well managed, daily builds can quickly outrun the testing capabilities, potentially causing testing to lag considerably behind the build process. Without the testing, ensuring that quality code is maintained throughout the development cycle becomes more difficult. The Agile process posted on the Security Development Lifecycle website specifies that this activity should be "time boxed" rather than performed every sprint.
- Implementing the SDL will help make utilization and conformance with the process recommendations of ISO/IEC 27034-1 easier. In our opinion, it may be the best enabling tool for conforming to ISO/IEC 27034-1.
- Incorporating the Microsoft SDL into your SDLC saves both development time and cost, while also reducing the security flaws in applications. It is a win-win-win, as it can be justified based on all three of these measurements.
- Microsoft has a wealth of information available at their SDL web portal, at http://www.microsoft.com/security/sdl/default.aspx

Appendix A – Security Development Lifecycle Overview

The contents of this appendix were provided by Microsoft.

Security Development Lifecycle Overview

The Security Development Lifecycle (SDL) is a security assurance process that is focused on software development. Combining a holistic and practical approach, the SDL introduces security and privacy throughout all phases of the development process, with the goal of reducing the number and severity of vulnerabilities in software.

The Microsoft SDL is based on three core concepts—*education*, *continuous process improvemen*t, and *accountability*. Ongoing education and training within a software development group is critical. The appropriate investment in knowledge transfer helps organizations react appropriately to changes in technology and the threat landscape. Because security risks are not static, the SDL places heavy emphasis on understanding the cause and effect of security vulnerabilities and *requires* regular evaluation of SDL processes and introduction of changes in response to new technology advancements or new threats. Data is collected to verify completion of security training, in-process metrics are used to confirm process compliance, and post-release metrics help guide future changes. Finally, the SDL requires the archival of all data necessary to service an application in a crisis. When this archived data is paired with detailed security response and communication plans, an organization can provide concise and cogent guidance to all parties affected by a security incident.

The SDL Process

Any software development organization, regardless of development methodology, can adopt the SDL process to integrate end-to-end security best practices.

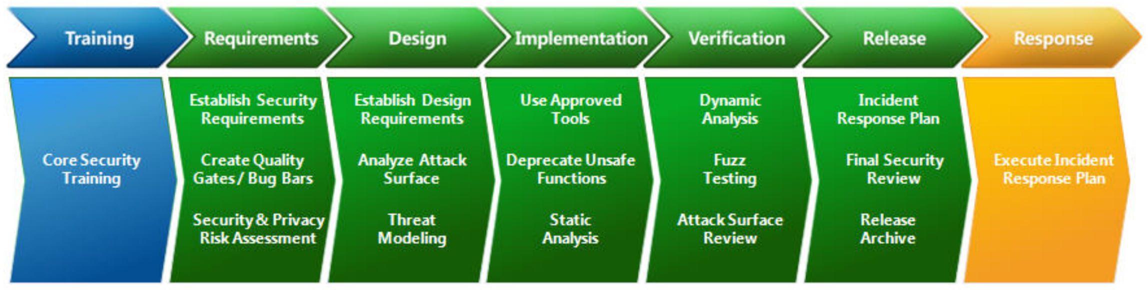

Figure 2: The SDL Process

It is important to notice that the five core phases roughly correspond to the phases within the traditional software development lifecycle:

- Requirements
- Design
- Implementation
- Verification
- Release and response

The SDL integrates effective security practices into each phase of the software development lifecycle to improve awareness of security risk and realize time and cost-saving benefits from discovering and eliminating security issues early in the development process.

1.0 SDL Security Training

1.1 Complete Core Security Training

All members of a software development team must receive appropriate training to stay informed about security basics and recent trends in security and privacy. The SDL applies this requirement to the entire organization to verify that security training is provided for everyone. Individuals in technical roles (developers, testers, and program managers) that are directly involved with the development of software programs must attend at least one unique security training class each year.

Basic software security training should cover foundational concepts such as:

Secure design

- Attack surface reduction
- Defense in depth
- Principle of least privilege
- Secure defaults

Threat modeling

- Overview of threat modeling
- Design implications of a threat model
- Coding constraints based on a threat model

Secure coding

- Buffer overruns (for applications using C and C++)
- Integer arithmetic errors (for applications using C and C++)
- Cross-site scripting (for managed code and web applications)
- SQL injection (for managed code and web applications)
- Weak cryptography

Security testing

- Differences between security testing and functional testing
- Risk assessment
- Security testing methods

Privacy

- Types of privacy-sensitive data
- Privacy design best practices
- Risk assessment
- Privacy development best practices
- Privacy testing best practices

2.0 Requirements Practices

2.1 Establish Security Requirements

The need to consider security and privacy "up front" is a fundamental aspect of secure system development. The initial planning stages are the optimal point to define trustworthiness requirements for a software project. This early definition of requirements allows development teams to identify key milestones and deliverables, and permits the integration of security and privacy that minimizes disruption to plans and schedules.

Create a basic risk questionnaire to verify whether the product should be subject to the SDL. At a minimum, products that meet the following criteria should follow a SDL process:

- Any product that is commonly used or deployed within a business (e.g. email or database servers).

- Any product that regularly stores, processes, or communicates personally identifiable information (PII) such as financial, medical, or sensitive customer information.
- Any online products or services that target or are attractive to children.
- Any product that regularly touches or listens on the Internet.
- Any product that automatically downloads updates.

If the results of this questionnaire show that the product should apply the SDL, begin building baseline security requirements from the content of the questionnaire.

Identify a security advisor to serve as the organization's first point of contact for security support and additional resources. This advisor should be responsible for defining the overall security policy and maintaining awareness of new threats or industry developments that may affect the security of the products or organization. In addition, identify the team or individual that is responsible for tracking and managing security for the product. This team or individual is not solely responsibility for addressing security in a software release, but this team or individual is responsible for coordinating and communicating the status of any security issues in the product. In smaller product groups, a single person can fill these roles.

It is important to establish the minimum design security requirements for the application that reflect how it will run in its planned operational environment. The security advisor, partnered with the product team security owner, should work with all disciplines to ensure security requirements are defined and agreed to early across the development organization. Once these requirements are established, identify and deploy a centralized security vulnerability work item tracking system that allows assigning, sorting, filtering, and tracking completion of security related bugs, work items, or tasks. The ability to track security work items is a critical piece in validating completion and generating data that demonstrates the effectiveness of establishing an SDL.

2.2 Create Quality Gates and Bug Bars

Quality gates and bug bars establish minimum acceptable levels of security and privacy quality. Defining these criteria at the start of a project improves the understanding of risks associated with security issues and enables teams to identify and avoid or fix security bugs during development. Establishing clear requirements early can improve engineering efficiencies in creating and executing quality assurance (QA) and test plans. A project team should define quality gates (for example, all compiler warnings must be triaged and fixed prior to code check-in) for each development phase, and then have them approved by the security advisor, who may add project-specific clarifications and more stringent security requirements as appropriate. The project team must illustrate compliance with the negotiated quality gates in order to complete the Final Security Review (FSR) before release.

A defined process should regulate the approval of exceptions to the quality gates and bug bars throughout the lifecycle of a project. This exception process should require approval from both product team management and security experts who understand any potential risks associated with a security exception and can make plans for mitigation in both incident response planning and future product cycles.

2.3 Perform Security Risk Assessment

Companies that perform in-house software development must include the threats and vulnerabilities associated with the software they develop in this risk assessment. These assessments should include some form of the following information:

- (Security) Which portions of the project require threat models before release?
- (Security) Which portions of the project require security design reviews before release?
- (Security) Which portions of the project (if any) require penetration testing by an organization that specializes in application security and is external to the project team?
- (Security) Are there any additional testing or analysis requirements the security advisor deems necessary to mitigate security risks?
- (Security) What is the specific scope of the fuzz testing requirements?
- (Compliance) What impact will compliance have on the product? Use your own framework to measure the impact of compliance. The following guidelines are provided as a beginning framework for a credit card processing example:
  - If the feature, product, or services stores *sensitive authentication data* (see definition), it is high risk.
  - If the feature, product, or service stores, processes, or transmits payment card data, (including only the Primary Account Number, cardholder name, expiration code, or service code), it is medium risk.
  - If the feature, product, or services does not store, process, or transmit any cardholder data or payment card data, it is low risk.

3.0 Design Practices

3.1 Establish Security Design Requirements

Establishing security design requirements involves a number of specific actions. These required activities include creating and reviewing security specifications for high-risk features, as well as defining secure coding techniques for developers. The results from these activities should be documented as the product's security design requirements.

All design specifications should describe how to securely implement all functionality provided by a given feature or function. It is a good practice to validate design specifications against the application's functional specification. The functional specification should:

- Accurately and completely describe the intended use of a feature or function.
- Describe how to deploy the feature or function in a secure fashion.
- Describe whether the feature or function will touch payment card data.

A key to PCI compliance is tying change control (Requirement 6.4) to data classification. This change control provides an archived record for developers to review application changes that impact controlled data such as payment card data.

Secure cryptographic design is a critical piece of both Design Phase SDL practices and PCI DSS compliance. Satisfying the minimal cryptographic design requirements established when creating product security requirements should be a priority. The SDL cryptographic requirements at a high level are:

- Use AES for symmetric encryption/decryption.
- Use 128-bit or better symmetric keys.
- Use RSA for asymmetric encryption/decryption and signatures.
- Use 1024-bit or better RSA keys.
- Use SHA-256 or better for hashing and message authentication codes

For additional details on this requirement, review the online SDL Process Guidance available at http://www.microsoft.com/security/sdl/discover/design.aspx.

3.2 Analyze Attack Surface

Attack surface reduction is a means of reducing risk by giving attackers less opportunity (surface) to exploit a potential vulnerability. Attack surface reduction may include shutting off or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible. At a minimum, attack surface reduction should include the following:

- Use Code Access Security (CAS) correctly. When developing with managed code, use strong-named assemblies and request minimal permission. When using strong-named assemblies, do not use Allow Partially Trusted Caller Attribute (APTCA) unless a security review approved use of the assembly.
- Manage firewall exceptions carefully. Be logical and consistent when making firewall exceptions. Any product or component that requires changes to the host firewall settings must adhere to the requirements that are outlined in the "Policy for Managing Firewall Configurations" document, available at http://msdn.microsoft.com/en-us/library/cc307394.aspx.
- Verify that the application runs correctly for users without administrator privileges. This restriction reduces the likelihood that a residual vulnerability in an application can be exploited to assume complete control of the underlying system.

3.3 Complete Threat Models

Use threat modeling for features or systems that were identified as having known security risk or the potential for risk during the Requirements Phase *Security Risk Assessment*. Threat modeling is a practice that allows development teams to consider, document, and discuss the security implications of designs in a planned operational environment. Threat modeling also allows consideration of security issues at the component or application level. It is a team exercise, encompassing program/project managers, developers, and testers, and represents the primary security analysis task performed during the software design stage. Threat modeling activities include:

- Complete threat models for all functionality identified as having known security risk or the potential for risk during the Requirements Phase *Security Risk Assessment*. Threat models typically must consider the following areas:
  - All projects—all code exposed on the attack surface and all code written by or licensed from a third party.
  - New projects—all features and functionality.
  - Updated versions of existing projects—new features or functionality added in the updated version.
- Verify that all threat models meet minimal quality requirements.
- Confirm that all threat models contain data flow diagrams, assets, vulnerabilities, and mitigations.

- Employ threat modeling using STRIDE[1]. To follow STRIDE, decompose the system into components, analyze each component for threats, and propose mitigations for each threat.
- Use tools such as the Microsoft SDL Threat Modeling Tool, a whiteboard or hand-drawn exercise combined with a thorough documentation of the results, or even using the SDL Elevation of Privilege threat modeling card game to perform threat modeling.
- Ensure that all threat models and referenced mitigations are approved by at least one security expert, one developer, one tester, and one program manager. Ask architects, developers, testers, program managers, and others who understand the software to contribute to threat models and to review them. Solicit broad input and review to verify that the threat models are as comprehensive as possible.

Threat model data and all associated documentation (functional and design specifications) should be stored by the product team to enable review of the threat models during the Verification Phase.

4.0 Implementation Practices

4.1 Use Approved Tools

All development teams should define and publish a list of approved tools and their associated security checks, such as compiler/linker options and warnings. The security advisor for the project team should approve this list. Development teams should use the latest versions of approved tools to take advantage of new security analysis functionality and protections.

4.2 Deprecate Unsafe Functions

Project teams should analyze all functions and APIs used in conjunction with a software development project and prohibit those that are determined to be unsafe. Once the banned API list is determined, project teams should use header files (such as banned.h and strsafe.h), newer compilers, or code scanning tools to check code (including legacy code where appropriate) for the existence of banned functions, and replace those banned functions with safer alternatives.

[1] STRIDE (Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, Elevation of privilege.)

4.3 Perform Static Analysis

Project teams should perform static analysis of source code. Static analysis of source code provides a tool-based scalable capability for security code review and can help verify the use of secure coding practices. Static code analysis by itself is generally insufficient to replace a manual code review for high-risk components. The security team and security advisors should be aware of the strengths and weaknesses of static analysis tools and be prepared to augment static analysis tools with other tools or human review as appropriate.

5.0 Verification Practices

5.1 Perform Dynamic Code Analysis

Run-time verification of software programs is necessary to verify that a program's functionality works as designed. This verification task should specify tools that monitor application behavior for memory corruption, user privilege issues, and other critical security problems. The SDL process uses run-time tools, along with other techniques such as fuzz testing, to achieve desired levels of security test coverage.

5.2 Perform Fuzz Testing

Fuzz testing is a specialized form of dynamic analysis used to induce program failure by deliberately introducing malformed or random data to an application. The strategy for fuzz testing should be derived from the intended use of the application and the functional and design specifications for the application. The security advisor may require additional fuzz tests or increases in the scope and duration of fuzz testing.

5.3 Conduct Attack Surface Review

It is common for an application to deviate significantly from the functional and design specifications created during the requirements and design phases of a software development project. Therefore, it is critical to re-review the threat models and attack surface of an application when it is code complete, to account for any design or implementation changes to the system and verify that mitigations are in place for any new attack vectors created.

In addition, review all security bugs identified against the quality gates and bug bars established in the Requirements Practices of the project to verify that the security requirements were achieved and the potential attack surface from exceptions is understood.

6.0 Release Practices

6.1 Create an Incident Response Plan

Every application, whether host- or web-based, should be supported by an incident response plan. Even programs with no known vulnerabilities at the time of release can be subject to new threats that emerge over time. The incident response plan should include:

- A contact list that identifies a sustained engineering team, or if the development group is too small to have these resources, a list of the appropriate engineering, marketing, communications, and management staff to act as points of first contact in a security emergency.
- On-call contacts with decision-making authority available 24 hours a day, seven days a week.
- Security servicing plans (escalation procedures) for code inherited from other groups within the organization.
- Security servicing plans (escalation procedures) for licensed third-party code, including file names, versions, source code, third-party contact information, and contractual permission to make changes (if appropriate).

6.2 Perform a Final Security Review

The Final Security Review (FSR) is a deliberate examination of all the security activities performed on a software application prior to release. The FSR is not a "penetrate-and-patch" exercise, nor is it a chance to perform security activities that were previously ignored or forgotten during the project. The FSR usually includes an examination of threat models, exception requests, tool output, and performance against the previously determined quality gates or bug bars. The FSR results in one of two different outcomes:

- **Passed FSR.** All security and privacy issues identified by the FSR process are fixed or mitigated.
- **Passed FSR with exceptions.** All security and privacy issues identified by the FSR process are fixed or mitigated and/or all exceptions are satisfactorily resolved. Those issues that cannot be addressed (for example, vulnerabilities posed by legacy "design-level" issues) are logged and corrected in the next release. If there's an exception, it must be reviewed by product team and security advisor. If the security advisor in partnership with the product team cannot reach an acceptable compromise, the security advisor cannot approve the project for release. Teams must either address whatever SDL requirements that they can prior to launch or escalate to executive management for a decision.

6.3 Archive All Release Data

Software release must be conditional on completion of the SDL process. The security advisor assigned to the release must certify that the project team has satisfied security requirements.

Archiving all pertinent information and data for reference during the Response Phase improves the speed and quality of response during incident response or post-release servicing of the software. Having all of the following items archived and available for reference and reuse equips a team with the full set of information they need to address security incidents, project post-mortems, and planning for next-version training and requirements:

- Feature specifications
- Source code, binaries, and private symbols
- Threat models
- Test cases
- Other related product documentation
- Emergency response plans
- License and servicing terms for any third-party software

Appendix B – ISO/IEC 27034-1

ISO/IEC 27034-1 Information technology — Security techniques — Application security (part 1 published, parts 2, 5 and 6 in DRAFT, parts 3 & 4 no text available)

ISO/IEC 27034-1 offers guidance on information security to those specifying, designing/programming or procuring, implementing and using application systems. In other words, addressing business and IT managers, developers and auditors, and ultimately the end-users of application systems. The aim is to ensure that computer applications deliver the desired/necessary level of security in support of the organization's Information Security Management System.

The multi-part standard provides guidance on specifying, designing/selecting and implementing information security controls through a set of **processes** integrated throughout an organization's Systems Development Life Cycle/s (SDLC). It is process-oriented.

It covers software applications developed internally, by external acquisition, outsourcing/offshoring or through hybrid approaches.

It addresses all aspects from determining information security requirements, to protecting information accessed by an application as well as preventing unauthorized use and/or actions of an application.

The standard is SDLC-method-agnostic: it does not mandate one or more specific development methods, approaches or stages but is written in a general manner to be applicable to them all. In this way, it complements other systems development standards and methods without conflicting with them.

ISO/IEC 27034-1:2011 — Information technology — Security techniques — Application security — Overview and concepts

- As with other multipartite ISO27k standards, the first part sets the scene for the remainder, providing a general introduction and outlining the remaining parts.
- ~80 pages long, with quite a bit of detail.
- States explicitly that this is not a software application development standard, an application project management standard, nor a software development cycle

standard. Its purpose is to provide general guidance that will be supported, in turn, by more detailed methods and standards in those areas.

- Explicitly takes a process approach to specifying, designing, developing, testing, implementing, and maintaining security functions and controls in application systems. For instance, it defines application security not as the state of security of an application system, but as "a process an organization can perform for applying controls and measurements to its applications in order to manage the risk of using them."
- Uses the concept of defining a Targeted Level of Trust (similar to a security plan) for an application, designing and building the application to meet it, and then validating the application against it.
- Draws on concepts such as auditing and certification of application systems similar in style to the Common Criteria and similar schemes primarily used for government and military systems. The text tends to emphasize deliberate threats arising from external adversaries implying the importance of confidentiality controls, arguably downplaying insider and accidental threats and the need for integrity and availability controls, but the process described ostensibly takes account of the full spectrum of security risks and controls.
- **Status**: part 1 was **published in 2011** and is available for CHF172 from the ISO/IEC webstore.