Microsoft

Microsoft®
Security Development Lifecycle

Simplified Implementation of the Microsoft SDL

Updated November 4, 2010

For the latest information, please see http://www.microsoft.com/sdl.

Contents

Introduction

The purpose of this paper is to illustrate the core concepts of the Microsoft Security Development Lifecycle (SDL) and to discuss the individual security activities that should be performed in order to claim compliance with the SDL process.

This 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.

The process outlined in this paper sets a *minimum* threshold for SDL compliance. That said, organizations aren't uniform – development teams should apply the SDL in a way that is suitable to the human talent and resources available, but doesn't compromise organizational security goals.

It is important to note that this document focuses solely on software development security methodologies used for building software applications and online services for external or internal release. There are other methodologies within an organization (such as IT security practices) that focus on "operational" security threats; while the groups that administer these processes may be bound by technology and regulatory contexts similar to those that bind software development groups, they generally do not create application software intended for use in environments where there is meaningful security risk.

Wherever possible, references to public sources of information are provided. Web links to specific discussions of processes, tools, and other ancillary information are included throughout this document.

About the Microsoft Security Development Lifecycle

The Security Development Lifecycle (SDL) is a security assurance process that is focused on software development. As a company-wide initiative and a mandatory policy since 2004, the SDL has played a critical role in embedding security and privacy in software and culture at Microsoft. Combining a holistic and practical approach, the SDL aims to reduce the number and severity of vulnerabilities in software. The SDL introduces security and privacy throughout all phases of the development process.

The Microsoft SDL is based on three core concepts—*education, continuous process improvement, and accountability.* The ongoing education and training of technical job roles within a software development group is critical. The appropriate investment in knowledge transfer helps organizations to react appropriately to changes in technology and the threat landscape. Because security risk is 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 assess training effectiveness, 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 paired with detailed security response and communication plans, an organization can provide concise and cogent guidance to all affected parties.

Secure Development at Microsoft

This document focuses on the application of the Microsoft SDL to development practices commonly in use by groups external to Microsoft. There is a strong correlation between this paper and the published process that is applied in the development of Microsoft products and services. It should, however, be noted that Microsoft expands on the core concepts discussed in this paper, and applies the SDL as a process that reflects the business and technical contexts of the organization. The SDL as practiced at Microsoft has evolved from the core concepts discussed in this paper and has been applied to hundreds of Microsoft products running on multiple operating systems and hardware platforms. These same core concepts are used by Microsoft teams in over 100 countries worldwide to meet the unique security and privacy challenges that arise from such a broad technology portfolio.

The Microsoft SDL process outlined in this paper focuses on applying proven security practices at distinct points in the software development life cycle. This process is technology-agnostic and not limited to large enterprises, making it easy to apply in organizations of any size.

Microsoft strongly believes in security and privacy process transparency. In response to user feedback asking for more details about how Microsoft applies the SDL to help secure its products and services, a detailed account of the Microsoft SDL process has been published at www.microsoft.com/sdl.

Microsoft SDL Optimization Model

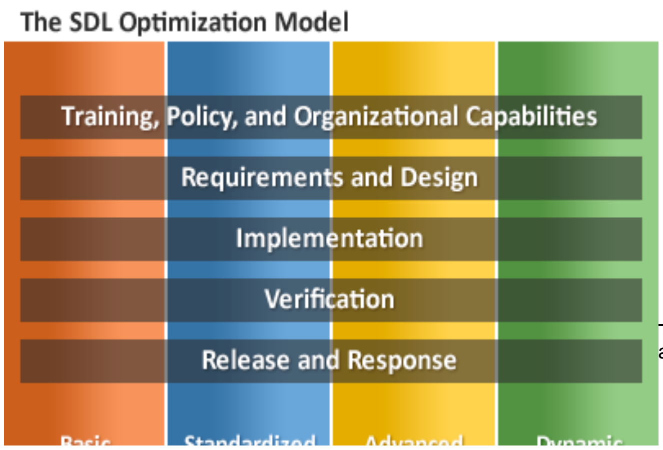
Integration of secure development concepts into an existing development process can be intimidating and costly if done improperly. Success or failure often hinges on variables such as organizational size, resources (time, talent, and budgets), and support from the executive suite. The impact of these intangibles can be controlled by understanding the elements of good security development practices and establishing implementation priorities based on the maturity level of the development team. Microsoft has created the SDL Optimization Model to help address these issues.

The SDL Optimization Model is structured around five capability areas that roughly correspond to the phases within the software development life cycle:

- Training, policy, and organizational capabilities
- Requirements and design
- Implementation
- Verification
- Release and response

Additionally, the SDL Optimization Model defines four levels of maturity for the practices and capabilities in these areas—Basic, Standardized, Advanced, and Dynamic.

The SDL Optimization Model starts at the Basic level of maturity, with little or no process, training, and tooling in place, and progresses to the Dynamic level, which is characterized by complete SDL compliance across an entire organization. Complete SDL compliance includes efficient and effective processes, highly trained individuals, specialized tooling, and strong accountability to parties both internal and external to the organization.



This paper focuses on the tasks and processes necessary to achieve the "advanced" maturity level.

That is, the point where an organization demonstrating competence in each of the five capability areas mentioned earlier can reasonably claim to follow the practices of the Microsoft SDL.

In contrast to other software maturity models, the Microsoft SDL Optimization Model focuses strictly on development process 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.

SDL Applicability

ıre 1. SDL Optimization Model with capability and maturity levels

It is vitally important to set clear expectations for an organization about the types of projects that will be subject to the controls imposed by the Microsoft SDL. Practical experience suggests that applications exhibiting one or more of the following characteristics should be subject to the SDL:

- Deployed in a business or enterprise environment
- Processes personally identifiable information (PII) or other sensitive information
- Communicates regularly over the Internet or other networks

Given the pervasiveness of computing technologies and the evolving threat environment, it may be easier to identify application development projects that *aren't* subject to security controls like those present in the SDL.

Roles, Responsibilities, and Qualifications of Security Personnel

The SDL includes general criteria and job descriptions for security and privacy roles. These roles are filled during the Requirements Phase of the SDL process. These roles are consultative in nature, and provide the organizational structure necessary to identify, catalog, and mitigate security and privacy issues present in a software development project. These roles include:

- **Reviewer/Advisory Roles.** These roles are designed to provide project security and privacy oversight and have the authority to accept or reject security and privacy plans from a project team.
    - *Security Advisor/Privacy Advisor.* These roles are filled by subject-matter experts (SMEs) from outside the project team. The role can either be filled by a qualified member of an independent, centralized group within the organization specifically chartered for reviews of this type, or by an expert external to the organization. The person chosen for this task must fill two sub-roles:
        - Auditor. This individual must monitor each phase of the software development process and attest to successful completion of each security requirement. The advisor must have the freedom to attest to compliance (or non-compliance) with security and privacy requirements without interference from the project team.
        - Expert. The person chosen for the advisor role must possess verifiable subject-matter expertise in security.
    - *Combination of Advisory Roles.* The role of security advisor may be combined with the role of privacy advisor if an individual with the appropriate skills and experience can be identified.
- **Team Champions.** The team champion roles should be filled by SMEs from the project team. These roles are responsible for the negotiation, acceptance, and tracking of minimum security and privacy requirements and maintaining clear lines of communication with advisors and decision makers during a software development project.

A centralized, internal advisory group is preferable; it provides organizational context and process knowledge beyond the capability of external experts.

- *Security Champion/Privacy Champion.* This individual (or group of individuals) does not have sole responsibility for ensuring that a software release has addressed all security issues, but is responsible for coordinating and tracking security issues for the project. This role also is responsible for reporting status to the security advisor and to other relevant parties (for example, development and test leads) on the project team.

- *Combination of Roles.* As with the security and privacy advisor role, the responsibilities vested in the champion role may be combined if an individual with the appropriate skills and experience can be identified.

Simplified SDL Security Activities

Simply put, the Microsoft SDL is a collection of mandatory security activities, presented in the order they should occur and grouped by the phases of the traditional software development life cycle (SDLC). Many of the activities discussed would provide *some* degree of security benefit if implemented on a standalone basis. However, practical experience at Microsoft has shown that security activities executed as part of a software development process lead to greater security gains than activities implemented piecemeal or in an ad-hoc fashion.

Focus should be placed on the accuracy of the output at each stage. Fancy reports with incomplete or incorrect information are a waste of time and resources.

Optional security activities may be added at the discretion of the project team or the security advisor to achieve desired security and privacy objectives. In the interest of brevity, detailed discussion of each security activity is omitted.

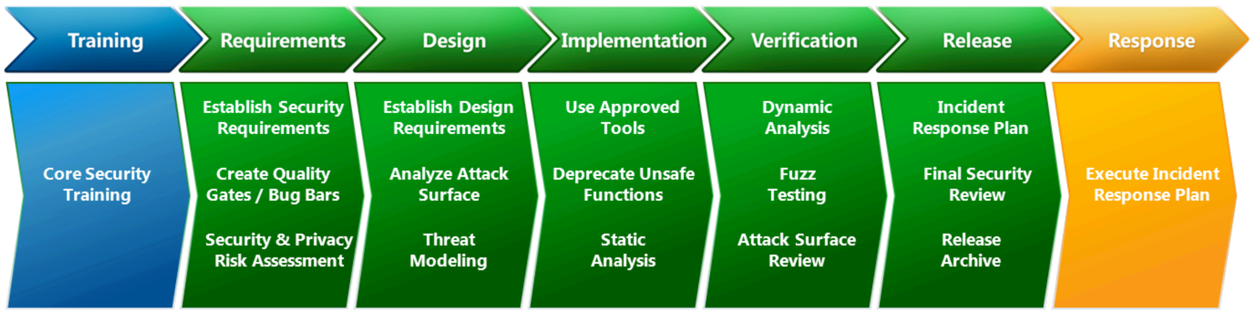


Figure 2: The Microsoft Security Development Lifecycle - Simplified

A fundamental concept to note is that an organization should focus on the quality and completeness of the output produced at each phase. A certain degree of security-process sophistication is expected of organizations operating at the Advanced and Dynamic levels of the SDL Optimization Model. That being said, it makes no difference if, for example, a threat model is produced as the result of a whiteboard session with the development team, is written out as a narrative in a Microsoft Word document, or is produced with the use of a specialized tool, such as the SDL Threat Modeling Tool. The SDL process does benefit from investments in effective tools and automation, but the real value lies in comprehensive and accurate results.

For ease of review, a detailed diagram that illustrates the SDL process flow can be found in Appendix A. This diagram is a visualization of the security activities used on a hypothetical project, ranging from training employees to application release. This example includes both mandatory and optional tasks.

Mandatory Security Activities

If a software development project is determined to be subject to the SDL (see the SDL Applicability section), the development team must successfully complete sixteen mandatory security activities to comply with the Microsoft SDL process. These mandatory activities have been acknowledged as effective by security and privacy experts, and are constantly reviewed for effectiveness as part of a rigorous annual evaluation process. As mentioned previously, development teams should retain the flexibility to specify other security activities as necessary, but the list of "must-complete" practices should always include the sixteen included in this document.

Pre-SDL Requirements: Security Training

SDL Practice 1: Training Requirements

All members of a software development team must receive appropriate training to stay informed about security basics and recent trends in security and privacy. 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, including the following topics:
  - Attack surface reduction
  - Defense in depth
  - Principle of least privilege
  - Secure defaults
- Threat modeling, including the following topics:
  - Overview of threat modeling
  - Design implications of a threat model
  - Coding constraints based on a threat model
- Secure coding, including the following topics:
  - 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, including the following topics:
  - Differences between security testing and functional testing
  - Risk assessment
  - Security testing methods
- Privacy, including the following topics:
  - Types of privacy-sensitive data
  - Privacy design best practices
  - Risk assessment
  - Privacy development best practices
  - Privacy testing best practices

The preceding training establishes an adequate knowledge baseline for technical personnel. As time and resources permit, training in advanced concepts may be necessary. Examples include, but are not limited to, the following:

- Advanced security design and architecture
- Trusted user interface design
- Security vulnerabilities in detail

- Implementing custom threat mitigations

Phase One: Requirements

SDL Practice 2: Security Requirements

The need to consider security and privacy “up front” is a fundamental aspect of secure system development. The optimal point to define trustworthiness requirements for a software project is during the initial planning stages. This early definition of requirements allows development teams to identify key milestones and deliverables, and permits the integration of security and privacy in a way that minimizes any disruption to plans and schedules. Security and privacy requirements analysis is performed at project inception and includes specification of minimum security requirements for the application as it is designed to run in its planned operational environment and specification and deployment of a security vulnerability/work item tracking system.

SDL Practice 3: Quality Gates/Bug Bars

Quality gates and *bug bars* are used to 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 fix security bugs during development. A project team must negotiate 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 also illustrate compliance with the negotiated quality gates in order to complete the Final Security Review (FSR).

A bug bar is a quality gate that applies to the entire software development project. It is used to define the severity thresholds of security vulnerabilities—for example, no known vulnerabilities in the application with a “critical” or “important” rating at time of release. The bug bar, once set, should never be relaxed. A dynamic bug bar is a moving target that is likely to be poorly understood within the development organization.

SDL Practice 4: Security and Privacy Risk Assessment

Security risk assessments (SRAs) and privacy risk assessments (PRAs) are mandatory processes that identify functional aspects of the software that require deep review. Such assessments must include the following information:

1. (Security) Which portions of the project will require threat models before release?
2. (Security) Which portions of the project will require security design reviews before release?
3. (Security) Which portions of the project (if any) will require penetration testing by a mutually agreed upon group that is external to the project team?
4. (Security) Are there any additional testing or analysis requirements the security advisor deems necessary to mitigate security risks?
5. (Security) What is the specific scope of the fuzz testing requirements?
6. (Privacy) What is the Privacy Impact Rating? The answer to this question is based on the following guidelines:

    - P1 High Privacy Risk. The feature, product, or service stores or transfers PII, changes settings or file type associations, or installs software.

    - P2 Moderate Privacy Risk. The sole behavior that affects privacy in the feature, product, or service is a one-time, user-initiated, anonymous data transfer (for example, the user clicks on a link and the software goes out to a Web site).

- P3 Low Privacy Risk. No behaviors exist within the feature, product, or service that affect privacy. No anonymous or personal data is transferred, no PII is stored on the machine, no settings are changed on the user's behalf, and no software is installed.

Phase Two: Design

SDL Practice 5: Design Requirements

The optimal time to influence a project's design trustworthiness is early in its life cycle. It is critically important to consider security and privacy concerns carefully during the design phase. Mitigation of security and privacy issues is much less expensive when performed during the opening stages of a project life cycle. Project teams should refrain from the practice of "bolting on" security and privacy features and mitigations near the end of a project's development.

In addition, it is crucially important for project teams to understand the distinction between "secure features" and "security features." It is quite possible to implement security features, which are in fact, insecure. *Secure features* are defined as features whose functionality is well engineered with respect to security, including rigorous validation of all data before processing or cryptographically robust implementation of libraries for cryptographic services. The term *security features* describes program functionality with security implications, such as Kerberos authentication or a firewall.

A formal exception or bug deferral method should be considered as part of any software development process. Many applications are based on legacy designs and code, so it may be necessary to defer certain security or privacy measures as a result of technical constraints.

The design requirements activity contains a number of required actions. Examples include the creation of security and privacy design specifications, specification review, and specification of minimal cryptographic design requirements. Design specifications should describe security or privacy features that will be directly exposed to users, such as those that require user authentication to access specific data or user consent before use of a high-risk privacy feature. In addition, all design specifications should describe how to securely implement all functionality provided by a given feature or function. It's 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.

SDL Practice 6: Attack Surface Reduction

Attack surface reduction is closely aligned with threat modeling, although it addresses security issues from a slightly different perspective. Attack surface reduction is a means of reducing risk by giving attackers less opportunity to exploit a potential weak spot or vulnerability. Attack surface reduction encompasses shutting off or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible.

SDL Practice 7: Threat Modeling

Threat modeling is used in environments where there is meaningful security risk. It is a practice that allows development teams to consider, document, and discuss the security implications of designs in the context of their planned operational environment and in a structured fashion. Threat modeling also allows consideration of security issues at the component or application level.

The preferred method for threat modeling is to use the SDL Threat Modeling Tool. The SDL Threat Modeling Tool is based on the STRIDE threat classification taxonomy.

Threat modeling is a team exercise, encompassing program/project managers, developers, and testers, and represents the primary security analysis task performed during the software design stage.

Phase Three: Implementation

SDL Practice 8: 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. This list should be approved by the security advisor for the project team. Generally speaking, development teams should strive to use the latest version of approved tools to take advantage of new security analysis functionality and protections.

SDL Practice 9: Deprecate Unsafe Functions

Many commonly used functions and APIs are not secure in the face of the current threat environment. Project teams should analyze all functions and APIs that will be used in conjunction with a software development project and prohibit those that are determined to be unsafe. Once the banned 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.

SDL Practice 10: Static Analysis

Project teams should perform static analysis of source code. Static analysis of source code provides a scalable capability for security code review and can help ensure that secure coding policies are being followed. Static code analysis by itself is generally insufficient to replace a manual code review. 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.

Generally speaking, development teams should decide the optimal frequency for performing static analysis – to balance productivity with adequate security coverage.

Phase Four: Verification

SDL Practice 11: Dynamic Program Analysis

Run-time verification of software programs is necessary to ensure 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 like AppVerifier, along with other techniques such as fuzz testing, to achieve desired levels of security test coverage.

SDL Practice 12: 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 fuzz testing strategy is 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.

SDL Practice 13: Threat Model and 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 threat models and attack surface measurement of a given application when it is code complete. This review ensures that any design or implementation changes to the system have been accounted for, and that any new attack vectors created as a result of the changes have been reviewed and mitigated.

Phase Five: Release

SDL Practice 14: Incident Response Plan

Every software release subject to the requirements of the SDL must include 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:

- An identified sustained engineering (SE) team, or if the team is too small to have SE resources, an emergency response plan (ERP) that identifies 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 that are available 24 hours a day, seven days a week.
- Security servicing plans for code inherited from other groups within the organization.
- Security servicing plans for licensed third-party code, including file names, versions, source code, third-party contact information, and contractual permission to make changes (if appropriate).

SDL Practice 15: 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 performed by the security advisor with assistance from the regular development staff and the security and privacy team leads. The FSR is not a "penetrate and patch" exercise, nor is it a chance to perform security activities that were previously ignored or forgotten. 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 three 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.
- **FSR with escalation.** If a team does not meet all SDL requirements and the security advisor and the product team cannot reach an acceptable compromise, the security advisor cannot approve the project, and the project cannot be released. Teams must either address whatever SDL requirements that they can prior to launch or escalate to executive management for a decision.

SDL Practice 16: Release/Archive

Software release to manufacturing (RTM) or release to Web (RTW) is conditional on completion of the SDL process. The security advisor assigned to the release must certify (using the FSR and other data) that the project team has satisfied security requirements. Similarly, for all products that have at least one component with a Privacy Impact Rating of P1, the project's privacy advisor must certify that the project team has satisfied the privacy requirements before the software can be shipped.

In addition, all pertinent information and data must be archived to allow for post-release servicing of the software. This includes all specifications, source code, binaries, private symbols, threat models, documentation, emergency response plans, license and servicing terms for any third-party software and any other data necessary to perform post-release servicing tasks.

Optional Security Activities

Optional security activities are generally performed when a software application is likely to be used in critical environments or scenarios. They are often specified by a security advisor as part of a negotiated set of additional requirements to ensure a greater level of security analysis for certain software components. The practices in this section provide examples of optional security tasks and should not be considered an exhaustive list.

Manual Code Review

Manual code review is an optional task in the SDL and is usually performed by highly skilled individuals on the application security team and/or the security advisor. While analysis tools can do much of the work of finding and flagging vulnerabilities, they are not perfect. As a result, manual code review is usually focused on the "critical" components of an application. Most often it is used where sensitive data, such as personally identifiable information (PII), is processed or stored. It is also used to examine other critical functionality such as cryptographic implementations.

Penetration Testing

Penetration testing is a white box security analysis of a software system performed by skilled security professionals simulating the actions of a hacker. The objective of a penetration test is to uncover potential vulnerabilities resulting from coding errors, system configuration faults, or other operational deployment weaknesses. Penetration tests are often performed in conjunction with automated and manual code reviews to provide a greater level of analysis than would ordinarily be possible.

Any issues identified during penetration testing must be addressed and resolved before the project is approved for release.

Vulnerability Analysis of Similar Applications

Many reputable sources of information about software vulnerabilities can be found on the Internet. In some cases, the analysis of vulnerabilities found in analogous software applications can shed light on potential design or implementation issues in software under development.

Other Process Requirements

Root Cause Analysis

While not traditionally a part of the software development process, root cause analysis plays an important part in ensuring software security. Upon discovery of a previously unknown vulnerability, an investigation should be performed to ascertain precisely where the security processes failed. These vulnerabilities can be attributed to a variety of causes, including human error, tool failure, and policy failure. The goal of root cause analysis is to understand the precise nature of the failure. This information helps to ensure that errors of the same type are accounted for in future revisions of the SDL.

Periodic Process Updates

Software threats are not static. As a result, the process used to secure software cannot be static. Organizations should take the knowledge learned from practices such as root cause analysis, policy changes, and improvements in technology and automation, and apply them to the SDL on a predictable schedule. Generally speaking, a yearly update schedule should suffice. The exception to this rule is when new, previously unknown vulnerability types are identified. This phenomenon requires immediate, out-of-cycle revision of the SDL to ensure proper mitigations are in place going forward.

Application Security Verification Process

Organizations developing secure software will naturally want a means to verify that the processes outlined in the Microsoft SDL have been followed. Access to centralized development and test data helps decision-making in a number of important scenarios, such as the Final Security Review, SDL requirement exception handling, and security audits. The process of verifying application security involves a number of different processes and actors:

- A specially designated application should be used to track compliance with the SDL. This application serves as the central repository for all SDL process artifacts, including (but not limited to) design and implementation notes, threat models, tool log uploads, and other process attestations. As with any critical application, it should use access controls to ensure:
    - Only authorized personnel can use the application.
    - Strong separation between roles. For example, a developer may be able to use the application and upload data, but should be prohibited from accessing functionality reserved for the security and privacy advisors, security team leads, and testers.

- The security and privacy team leads are responsible for ensuring that the data necessary for an objective judgment is properly categorized and entered into the tracking application.
- The information entered into the tracking application is used by the security and privacy advisors to provide the analysis framework for the Final Security Review.
- The security and privacy advisors are responsible for reviewing the data entered into the tracking application (including the FSR results and other additional security tasks assigned by the advisors) and certifying that all requirements are met and/or all exceptions are satisfactorily resolved.

This document focuses on the Advanced level of the SDL Optimization Model, where rudimentary tracking processes are (in most instances) insufficient to the task. However, organizations with less sophisticated processes or smaller resource pools—those who fit within the Basic or Standardized levels of the SDL Optimization Model—can likely make do with a simpler tracking process.

It is very important that the tracking and verification process accurately capture:

- The security and privacy requirements of the organization (for example, no known critical vulnerabilities at release).
- The functional and technical requirements of the application under development.
- The application's operational context.

For example, if a development team creates a process control application to run in a critical environment, the proper investment of time and resources must be allocated to the creation and maintenance of the tracking process to enable objective analyses by the organization's security and privacy principals, executive leadership, and relevant third parties such as compliance auditors or evaluators. Put differently, skimping on the tracking process inevitably leads to problems later, usually during an emergency. Ensure that reliable systems are in place to answer critical questions at critical times.

Conclusion

The goal of this paper is to provide a *simple* framework for the pragmatic inclusion of security practices in the software development process. It outlines a series of discrete, non-proprietary security development activities that when joined with effective process automation and solid policy guidance represent the steps necessary for an organization to objectively claim compliance with the Microsoft SDL as defined by the “Advanced” level in the SDL Optimization Model.

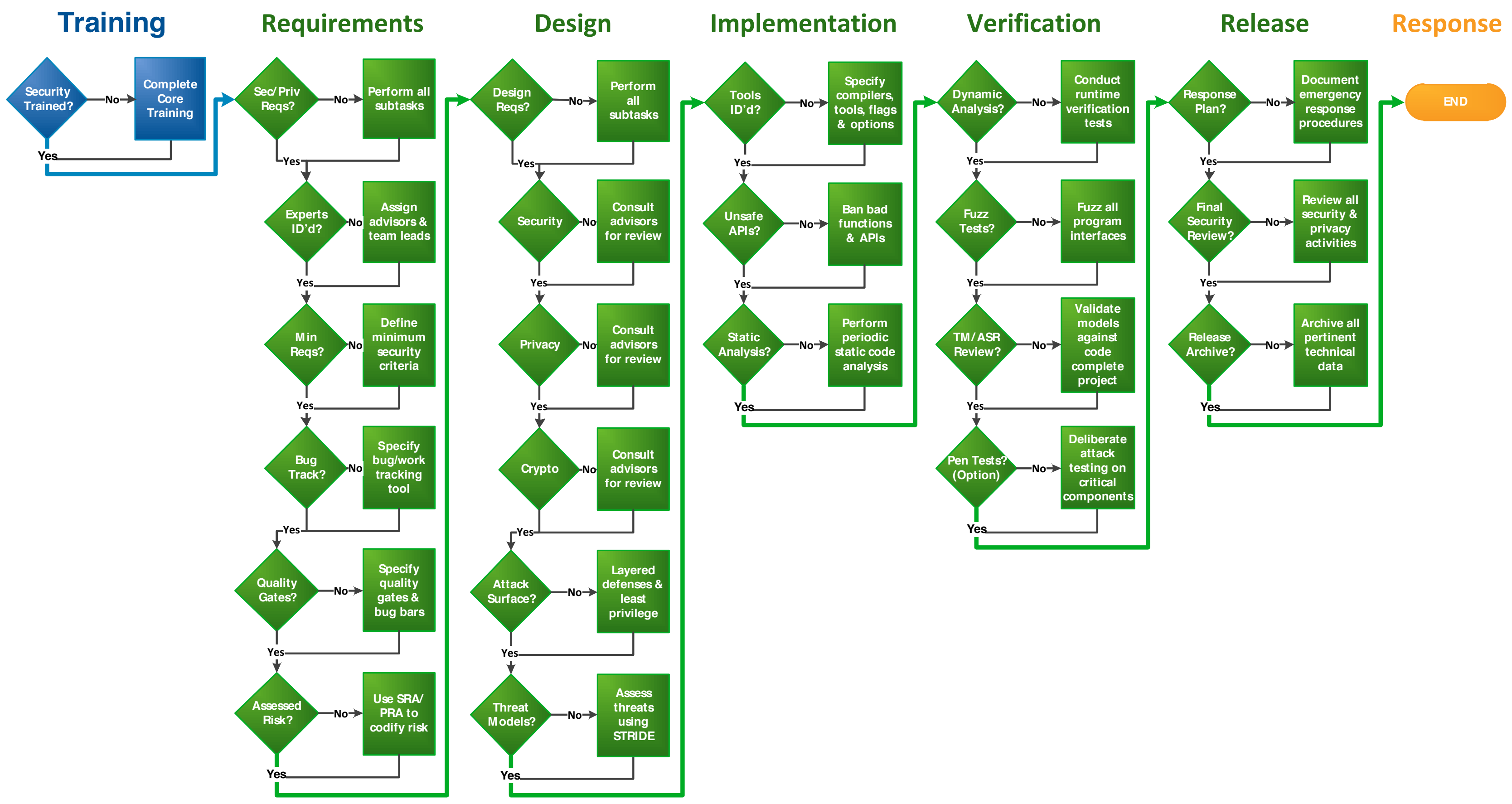
While the process outlined above sets a *minimum* threshold for SDL compliance, the SDL is not “one size fits all.” Development teams should use this document as a *guide* for implementing the SDL in a fashion appropriate to the time, resources and business practices of the organization.

The engineering concepts discussed are generally recognized as good security practices and not specific to Microsoft or the Windows platform. They are applicable to different operating systems, hardware platforms, and development methodologies. Even a piecemeal approach using a few of these techniques would likely have a beneficial influence on the security and privacy of an application under development.

It can be argued that even simple orchestration of security processes can lead to holistic improvements in security and privacy that transcend the value of individually performed tasks. However, the threats present in today’s computing environments have evolved from the early days of script kiddies seeking an ego boost to widespread organized financial crime, and more recently to new battlefields for nation-state warfare. This threat evolution argues strongly for something much more substantial than the typical piece-meal approach to software security.

The Microsoft SDL is a freely available process for improving software security and privacy. It has been applied to hundreds of software programs and hundreds of millions of lines of production code. The SDL consists of mandatory actions that follow the traditional software development process, yet it’s flexible enough to allow for the addition of other policies and techniques, thereby creating a software development methodology that is unique to an organization. The combination of process, training, and tools produces distinct benefits, such as increased predictability, technical competence, and more secure software, which translates into lower risk for both the organization and the software user.

Appendix A: SDL Process Illustration


Training
Requirements
Design
Implementation
Verification
Release
Response
Security Trained?
No
Complete Core Training
Yes
Sec/Priv Reqs?
Perform all subtasks
Experts ID'd?
Assign advisors & team leads
Min Reqs?
Define minimum security criteria
Bug Track?
Specify bug/work tracking tool
Quality Gates?
Specify quality gates & bug bars
Assessed Risk?
Use SRA/ PRA to codify risk
Design Reqs?
Perform all subtasks
Security
Consult advisors for review
Privacy
Consult advisors for review
Crypto
Consult advisors for review
Attack Surface?
Layered defenses & least privilege
Threat Models?
Assess threats using STRIDE
Tools ID'd?
Specify compilers, tools, flags & options
Unsafe APIs?
Ban bad functions & APIs
Static Analysis?
Perform periodic static code analysis
Dynamic Analysis?
Conduct runtime verification tests
Fuzz Tests?
Fuzz all program interfaces
TM/ASR Review?
Validate models against code complete project
Pen Tests? (Option)
Deliberate attack testing on critical components
Response Plan?
Document emergency response procedures
Final Security Review?
Review all security & privacy activities
Release Archive?
Archive all pertinent technical data
END