MICROSOFT CORPORATION

MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (SDL)

Version 3.2

The Microsoft Security Development Lifecycle (SDL) is an industry-leading software security assurance process. A Microsoft-wide initiative and a mandatory policy since 2004, the SDL has played a critical role in embedding security and privacy in Microsoft software and culture. Combining a holistic and practical approach, the SDL introduces security and privacy early and throughout all phases of the development process. It has led Microsoft to measurable and widely-recognized security improvements in flagship products such as Windows Vista and SQL Server. Microsoft is publishing the detailed SDL process guidance as part of its commitment to enable a more secure and trustworthy computing ecosystem.

The following documentation provides an in-depth description of the Microsoft SDL methodology and requirements. Proprietary technologies and resources that are only available internally at Microsoft have been omitted from these guidelines.

CONTENTS

INTRODUCTION

All software developers must address security threats. Computer users now require trustworthy and secure software, and developers who address security threats more effectively than others can gain competitive advantage in the marketplace. Also, an increased sense of social responsibility now compels developers to create secure software that requires fewer patches and less security management.

Privacy also demands attention. To ignore privacy concerns of users can invite blocked deployments, litigation, negative media coverage, and mistrust. Developers who protect privacy earn users' loyalty and distinguish themselves from their competitors.

Secure software development has three elements: best practices, process improvements, and metrics. This document focuses primarily on the first two elements, and metrics are derived from measuring how they are applied.

Microsoft has implemented a stringent software development process that focuses on these elements. The goal is to minimize security-related vulnerabilities in the design, code, and documentation and to detect and eliminate vulnerabilities as early as possible in the development lifecycle. These improvements reduce the number and severity of security vulnerabilities and improve the protection of users' privacy.

Secure software development is mandatory for software that is developed for the following uses:

- In a business environment
- To process personally identifiable information or other sensitive information
- To communicate regularly over the Internet or other networks

(For more specific definitions, see the "What Products and Services are Required to Adopt the Security Development Lifecycle Process?" section later in this Introduction.)

This document describes both required and recommended changes to software development tools and processes. These changes should be integrated into existing software development processes to facilitate best practices and achieve measurably improved security and privacy.

Note: *This document outlines the SDL process used by Microsoft product groups for application development. It has been modified slightly to remove references to internal Microsoft resources and to minimize Microsoft-specific jargon. We make no guarantees as to its applicability for all types of application development or for all development environments. Implementers should use common sense in choosing the portions of the SDL that make sense given existing resources and management support.*

THE TRADITIONAL MICROSOFT PRODUCT DEVELOPMENT PROCESS

In response to the Trustworthy Computing (TwC) directive of January 2002, many software development groups at Microsoft instigated *security pushes* to find ways to improve the security of existing code and one or two prior versions of the code. However, the reliable delivery of more secure software requires a comprehensive process, so Microsoft defined SD3+C (Secure by Design, Secure by Default, Secure in Deployment, and Communications) to help determine where security and privacy efforts are needed. The guiding principles for SD3+C are identified in the following subsections.

SECURE BY DESIGN

- **Secure architecture, design, and structure**. Developers consider security issues part of the basic architectural design of software development. They review detailed designs for possible security issues and design and develop mitigations for all threats.
- **Threat modeling and mitigation**. Threat models are created, and threat mitigations are present in all design and functional specifications.
- **Elimination of vulnerabilities**. No known security vulnerabilities that would present a significant risk to anticipated use of the software remain in the code after review. This review includes the use of analysis and testing tools to eliminate classes of vulnerabilities.
- **Improvements in security**. Less secure legacy protocols and code are deprecated, and, where possible, users are provided with secure alternatives that are consistent with industry standards.

SECURE BY DEFAULT

- **Least privilege**. All components run with the fewest possible permissions.
- **Defense in depth**. Components do not rely on a single threat mitigation solution that leaves customers exposed if it fails.
- **Conservative default settings**. The development team is aware of the attack surface for the product and minimizes it in the default configuration.
- **Avoidance of risky default changes**. Applications do not make any default changes to the operating system or security settings that reduce security for the host computer. In some cases, such as for security products (for example, Microsoft® Internet Security and Acceleration (ISA) Server), it is acceptable for a software program to strengthen (increase) security settings for the host computer. The most common violations of this principle are games that either open up firewall ports without informing the user or instruct users to do so without consideration of possible risks.
- **Less commonly used services off by default**. If fewer than 80 percent of a program's users use a feature, that feature should not be activated by default. Measuring 80 percent usage in a product is often difficult, because programs are designed for many different personas. It can be useful to consider whether a feature addresses a core/primary use scenario for all personas. If it does, the feature is sometimes referred to as a P1 feature.

SECURE IN DEPLOYMENT

- **Deployment guides**. Prescriptive deployment guides outline how to deploy each feature of a program securely, including providing users with information that enables them to assess the security risk of activating non-default options (and thereby increasing the attack surface).

- **Analysis and management tools**. Security analysis and management tools enable administrators to determine and configure the optimal security level for a software release. These tools include Microsoft Baseline Security Analyzer as well as Group Policy, through which you can manage all security-related configuration options.
- **Patch deployment tools**. Deployment tools are provided to aid in patch deployment.

COMMUNICATIONS

- **Security response**. Development teams respond promptly to reports of security vulnerabilities and communicate information about security updates.
- **Community engagement**. Development teams proactively engage with users to answer questions about security vulnerabilities, security updates, or changes in the security landscape.

An analogous concept to SD3+C for privacy is known as PD3+C. The guiding principles for PD3+C are:

PRIVACY BY DESIGN

- **Provide notice and consent**. Provide appropriate notice about data that is collected, stored, or shared so that users can make informed decisions about their personal information.
- **Enable user policy and control**. Enable parents to manage privacy settings for their children and enterprises to manage privacy settings for their employees.
- **Minimize data collection and sensitivity**. Collect the minimum amount of data that is required for a particular purpose, and use the least sensitive form of that data.
- **Protect the storage and transfer of data**. Encrypt personally identifiable information in transfer, limit access to stored data, and ensure that data usage complies with uses communicated to the customer.

PRIVACY BY DEFAULT

- **Ship with conservative default settings**. Obtain appropriate consent before collecting or transferring any data. To prevent unauthorized access, protect personal data stored in access control lists.

PRIVACY IN DEPLOYMENT

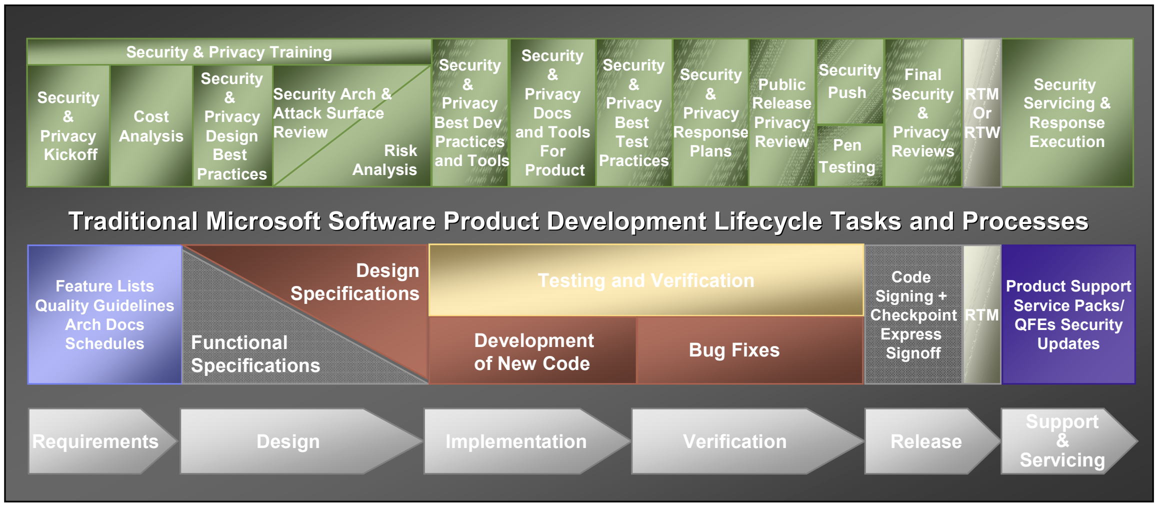
- **Publish deployment guides**. Disclose privacy mechanisms to enterprise customers so that they can establish internal privacy policies and maintain their customers’ and employees' privacy.

COMMUNICATIONS

- **Publish author-appropriate privacy disclosures**. Post privacy statements on appropriate Web sites.
- **Promote transparency**. Actively engaging mainstream and trade media outlets with white papers and other documentation can help reduce anxiety about high-risk features.
- **Establish a privacy response team**. Assign staff who are responsible for responding if a privacy incident or escalation occurs.

THE SECURITY DEVELOPMENT LIFECYCLE (SDL)

After you add steps to the development process for all elements of SD3+C, the secure software development process model looks like the one shown in the following figure:


Security & Privacy Training
Security & Privacy Kickoff
Cost Analysis
Security & Privacy Design Best Practices
Security Arch & Attack Surface Review
Risk Analysis
Security & Privacy Best Dev Practices and Tools
Security & Privacy Docs and Tools For Product
Security & Privacy Best Test Practices
Security & Privacy Response Plans
Public Release Privacy Review
Security Push
Pen Testing
Final Security & Privacy Reviews
RTM Or RTW
Security Servicing & Response Execution
Traditional Microsoft Software Product Development Lifecycle Tasks and Processes
Feature Lists Quality Guidelines Arch Docs Schedules
Design Specifications
Functional Specifications
Testing and Verification
Development of New Code
Bug Fixes
Code Signing + Checkpoint Express Signoff
RTM
Product Support Service Packs/ QFEs Security Updates
Requirements
Design
Implementation
Verification
Release
Support & Servicing

Process improvements are incremental and do not require radical changes in the development process. However, it is important to make improvements consistently across an organization.

The rest of this document describes each step of the process in detail.

WHAT PRODUCTS AND SERVICES ARE REQUIRED TO ADOPT THE SECURITY DEVELOPMENT LIFECYCLE PROCESS?

- Any software release that is commonly used or deployed within any organization, such as a business organization or a government or nonprofit agency.
- Any software release that regularly stores, processes, or communicates *personally identifiable information* (PII) or other sensitive information. Examples include financial or medical information.
- Any software product or service that targets or is attractive to children 13 years old and younger.
- Any software release that regularly connects to the Internet or other networks. Such software might be designed to connect in different ways, including:
  - **Always online**. Services provided by a product that involve a presence on the Internet (for example, Windows Messenger).
  - **Designed to be online**. Browser or mail applications that expose Internet functionality (for example, Microsoft Outlook® or Internet Explorer®).
  - **Exposed online**. Components that are routinely accessible through other products that interact with the Internet (for example, Microsoft ActiveX® controls or PC–based games with multiplayer online support).
- Any software release that automatically downloads updates

- Any software release that accepts and/or processes data from an unauthenticated source, including:
  - Callable interfaces that “listen”
  - Functionality that parses any unprotected file types should be limited to system administrators
- Any release that contains ActiveX controls
- Any release that contains COM controls

ARE “SERVICE RELEASES” REQUIRED TO ADOPT THE SECURITY DEVELOPMENT LIFECYCLE PROCESS?

Any external release of software that can be installed on a customer’s computer, regardless of operating system or platform, must comply with security and privacy policies as described in the Security Development Lifecycle. This Lifecycle applies to new products, service releases such as product service packs, feature packs, development kits, and resource kits. The terms *service pack* and *feature pack* might not always be used in the descriptive title of a release to customers, but the following definitions differentiate what constitutes a *new product* from a service release or feature pack.

- **New Product Releases** are either completely new products (version 1.0) or significant updates of existing products (for example, Microsoft Office 2003). A new product release always requires a customer to agree to a new software license, and typically involves new packaging.
- **Service Releases** include service packs or feature packs and resource kits.
- **Service Packs** are the means by which product updates are distributed. Service packs might contain updates for system reliability, program compatibility, security, or privacy. A service pack requires a previous version of a product before it can be installed and used. A service pack might not always be named as such; some products may refer to a service pack as a *service release*, *update*, or *refresh*.
- **Resource Kits** are collections of resources to help administrators streamline management tasks. A resource kit must be targeted at a single product release to be treated as a service release. If a resource kit is targeted at multiple products, or at multiple versions of a product, SDL requirements will apply to it as described above for a product release.
- **Development Kits** provide information, detailed architecture details, and tools to developers. A development kit must be targeted at a single product release to be treated as a service release. If a development kit is targeted at multiple products or at multiple versions of a product, SDL requirements from the corresponding product release will apply.

All software releases referenced in the "What Products and Services Are Required to Adopt the Security Development Lifecycle Process?" section must adopt SDL. However, current SDL requirements will be applied only to the new features in the service release and not retroactively to the entire product. Also, product teams are not required to change compiler versions or compile options in a service release.

HOW ARE NEW RECOMMENDATIONS AND NEW REQUIREMENTS ADDED TO THE SECURITY DEVELOPMENT LIFECYCLE PROCESS?

The Security Development Lifecycle consists of the proven best practices and tools that were successfully used to develop recent products. However, the area of security and privacy changes frequently, and the Security Development Lifecycle must continue to evolve and use new knowledge and tools to help build even more trusted products. But because product development teams must also have some visibility and predictability of security requirements in order to plan schedules, it is necessary to define how new recommendations and requirements are introduced, as well as when new requirements are added to the SDL.

New SDL recommendations may be added at any time, and they do not require immediate implementation by product teams.

New SDL requirements should be released and published at six-month intervals. New requirements will be finalized and published three months before the beginning of the next six-month interval for which they are required. For more information about how to hold teams accountable for requirements, see the following "How Are SDL Requirements Determined for a Specific Product Release?" section.

The list of required development tools (for example, compiler versions or updated security tools) will typically be the area of greatest interest, because of the potential impact on schedule and resources.

The following example timeline helps to illustrate this point:

- **October 1, 2006**. Publish updated requirements that will apply to all products registering after January 1, 2007.
- **January 1, 2007**. Updated requirements list takes effect.
- **April 1, 2007**. Publish updated requirements that will apply to all products registering after July 1, 2007.
- **July 1, 2007**. Updated requirements list takes effect.

HOW ARE SDL REQUIREMENTS DETERMINED FOR A SPECIFIC PRODUCT RELEASE?

A product release will be held accountable for the SDL requirements that are current on the day the product registers a request for SDL review. Product teams will refer to the SDL version numbers to determine the appropriate policies to follow. There are some caveats to this rule:

- **One-year cap**. At a minimum, a product must meet SDL requirements that are older than one year at the time of release to manufacture (RTM) or release to Web (RTW).
- **Multi-version limit**. If you register more than one version of your product to be released in succession, later versions will be held to the requirements that are in effect on the date the previous version was released.
- **Previous version**. All projects that are already registered before July 1 of a given year will be subject to the SDL requirements published on January 1 of the same year.
- **Threat evolution**. The security engineering team reserves the right to add new requirements at any time in response to the availability of high-impact tools or the evolution of new threats.

The following examples illustrate how SDL requirements are determined:

- If a product registered with the SDL team in H1CY05 (first half of calendar year 2005) but does not ship until H2CY06, it must meet all H2CY05 requirements.
- If a team registers two versions of a product to be released within a three-month period, the first version is subject to current requirements, but the second version is subject to the published requirements on the day the first version ships.

STAGE 0: EDUCATION AND AWARENESS

All members of software development teams should receive appropriate training to stay informed about security basics and recent trends in security and privacy. Individuals who develop software programs should attend at least one security training class each year. Security training can help ensure software is created with security and privacy in mind, and can also help development teams stay current on security issues. Project team members are strongly encouraged to seek *additional* security and privacy education that is appropriate to their needs and/or products.

A number of key knowledge concepts are important to successful software security. These concepts can be broadly categorized as either basic or advanced security knowledge. Each technical member of a project team (Developer, Tester, Program Manager) should be exposed to the knowledge concepts in the following subsections.

BASIC CONCEPTS

- **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 to a threat model
    - Coding to a threat model
    - Testing to a threat model
- **Secure Coding**, including the following topics:
    - Buffer overruns
    - Integer arithmetic errors
    - Cross site scripting
    - SQL injection
    - Weak cryptography
    - Managed code issues (Microsoft .NET/Java)
- **Security Testing**, including the following topics:
    - Security testing vs. Functional testing
    - Risk assessment
    - Test methodologies
    - Test automation

- **Privacy**, including the following topics:
  - Types of privacy data
  - Privacy design best practices
  - Risk analysis
  - Privacy development best practices
  - Privacy testing best practices

ADVANCED CONCEPTS

The preceding training concepts establish an adequate knowledge baseline for technical personnel. As time and resources permit, we recommend that you explore other advanced concepts. Examples include (but are not limited to):

- Security design and architecture
- User interface design
- Security bugs in detail
- Security response processes
- Implementing custom threat mitigations

SECURITY REQUIREMENTS

- All developers, testers, and program managers must complete at least one security training class each year. Individuals who have not taken a class in the basics of security design, development, and testing must do so.
- At least 80% of the project team staff, who work on products or services must be in compliance with the standards listed above before their product or service is released. Relevant managers must also be in compliance with these standards. We strongly encourage project teams to plan security training early in the development process, so that training can be completed as early as possible and have a maximum positive effect on the project's security.

SECURITY RECOMMENDATIONS

We recommend that staff who work in all disciplines read the following publications:

- *Writing Secure Code, Version 2* (ISBN: 0-7356-1722-8).
- *Uncover Security Design Flaws Using The STRIDE Approach* (ISBN: 0-7356-1991-3).

PRIVACY RECOMMENDATIONS

We recommend that staff who work in all disciplines read the following documents:

- Appendix A: Privacy at a Glance (Sample)
- *Microsoft Privacy Guidelines for Developing Software Products and Services*

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 5: Stage 0 – Education and Awareness
- *Privacy: What Developers and IT Professionals Should Know* (ISBN-10: 0-321-22409-4; ISBN-13: 978-0-321-22409-5)
- *The Protection of Information in Computer Systems*

- *Bell-LaPadula Model*
- *Biba Model*

STAGE 1: PROJECT INCEPTION

The need to consider security and privacy at a foundational level is a fundamental tenet of system development. The best opportunity to build trusted software is during the initial planning stages of a new release or a new version, because development teams can identify key objects and integrate security and privacy, which minimizes disruption to plans and schedules.

SECURITY REQUIREMENTS

- Develop and answer a short questionnaire to verify whether your development team is subject to Security Development Lifecycle (SDL) policies. The questionnaire has two possible outcomes:
    1. If the project is subject to SDL policies, it must be assigned a security advisor who serves as the point of contact for its Final Security Review (FSR). It is in the project team's interest to register promptly and establish the security requirements for which they will be held accountable. The team will also be asked some technical questions as part of a security risk assessment to help the security advisor identify potential security risks. (See "Stage 2: Cost Analysis" in this document.)
    2. If the project is not subject to SDL policies, it is not necessary to assign a security advisor and the release will be classified as exempt from SDL security requirements.
- Identify the team or individual that is responsible for tracking and managing security for the product. This team or individual does not have sole responsibility for ensuring that a software release is secure, but the team or individual is responsible for coordinating and communicating the status of any security issues. In smaller product groups, a single program manager might take this role.
- Ensure that bug reporting tools can track security bugs and that a database can be queried dynamically for all security bugs at any time. The purpose of this query is to examine unfixed security bugs in the FSR. The project's bug tracking system must accommodate the *bug bar* ranking value recorded with each bug.
- Define and document a project's security bug bar. This set of criteria establishes a minimum level of quality. Defining it at the start of a project improves understanding of risks associated with security issues and enables teams to identify and fix security bugs during development. The project team must negotiate a bug bar approved by the security advisor with project-specific clarifications and (as appropriate) more stringent security requirements specified by the security advisor. ***The bug bar must never be relaxed, though, even as a project's release date nears.*** Bug bar examples can be found in the "SDL Privacy Bug Bar and SDL Security Bug Bar (Samples)" (Appendices M and N respectively).

PRIVACY REQUIREMENTS

- Identify the privacy advisor who will serve as your team's first point of contact for privacy support and additional resources.
- Identify who on the team is responsible for privacy for a project. This person is typically called the *privacy lead* or, sometimes, the *privacy champion*.
- Define and document the project's privacy bug bar (see the preceding "Security Requirements" section).

SECURITY RECOMMENDATIONS

It is useful to create a security plan document during the design phase, to outline the processes and work items your team will follow to integrate security into their development process. The security plan should identify the timing and resource requirements that the Security Development Lifecycle prescribes for individual activities. These requirements should include:

- Team training
- Threat modeling
- Security push
- Final Security Review (FSR)

The security plan should reflect a development team's overall perspective on security goals, challenges, and plans. Security plans can change, but articulating one early helps ensure that no requirements are overlooked and avoid last-minute surprises. A sample security plan is included in Appendix O**.**

Consider using a tool to track security bugs by cause and effect. This information is very important to have later in a project. Ensure that the bug reporting tool you use includes fields with the STRIDE values in the following lists (definitions for these values are available in "Appendix B: Security Definitions for Vulnerability Work Item Tracking").

The tool's **Security Bug Effect** field should be set to one or more of the following STRIDE values:

- Not a Security Bug
- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation Disclosure
- **D**enial of Service
- **E**levation of Privilege
- Attack Surface Reduction

It is also important to use the **Security Bug Cause** field to log the cause of a vulnerability (this field should be mandatory if **Security Bug Effect** is anything other than *Not a Security Bug*).

The **Security Bug Cause** field should be set to one of the following values:

- Not a security bug
- Buffer overflow/underflow
- Arithmetic error (for example, integer overflow)
- SQL/Script injection
- Directory traversal
- Race condition
- Cross-site scripting
- Cryptographic weakness
- Weak authentication
- Weak authorization/Inappropriate permission or ACL (access control list)
- Ineffective secret hiding
- Unlimited resource consumption (Denial of Service, or DoS)
- Incorrect/No error messages
- Incorrect/No pathname canonicalization

- Other

Be sure to configure bug reporting tools correctly; limit access to bugs with security implications to the project team and security advisors only.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 6: Stage 1 – Project Inception

STAGE 2: COST ANALYSIS

Before you invest time in design and implementation, it is important to understand the costs and requirements involved in handling data with privacy considerations. Privacy risks increase development and support costs, so improving security and privacy can be consistent with business needs.

SECURITY REQUIREMENTS

A security risk assessment (SRA) is a mandatory exercise to identify functional aspects of the software that might require deep security review Given that program features and intended functionality might be different from project to project, it is wise to start with a simple SRA and expand it as necessary to meet the project scope.

Such assessments must include the following information:

- What portions of the project will require threat models before release.
- What portions of the project will require security design reviews before release.
- What portions of the project will require penetration testing (pen testing) by a mutually agreed upon group that is external to the project team. Any portion of the project that requires pen testing must resolve issues identified during pen testing before it is approved for release.
- Any additional testing or analysis requirements the security advisor deems necessary to mitigate security risks.
- Clarification of the specific scope of *fuzz testing* requirements. ("Stage 7: Verification Phase: Security and Privacy Testing" discusses fuzz testing.)

NOTE: SRA guidelines are discussed in Chapter 8 of The Security Development Lifecycle, along with a sample SRA on the DVD included with the book.

PRIVACY REQUIREMENTS

Complete the "Initial Assessment" section of "Appendix C: SDL Privacy Questionnaire". An initial assessment is a quick way to determine a project's Privacy Impact Rating and estimate how much work is necessary to comply with Microsoft Privacy Guidelines for Developing Software Products and Services.

The Privacy Impact Rating (P1, P2, or P3) measures the sensitivity of the data your software will process from a privacy point of view. More information about Privacy Impact Ratings can be found in Chapter 8 of The Security Development Lifecycle. General definitions of privacy impact are defined below:

- P1 – High Privacy Risk: The feature, product, or service stores or transfers Personally Identifiable Information (PII) or error reports; monitors the user with an ongoing transfer of anonymous data; 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 (e.g., the user clicks on a link and goes out to a website).

- 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.

Product teams must complete only the work that is relevant to their Privacy Impact Rating. Complete the initial assessment early in the product planning/requirements phase, before you write detailed specifications or code.

PRIVACY RECOMMENDATIONS

If your Privacy Impact Rating is P1 or P2, understand your obligations and try to reduce your risk. Early awareness of all the required steps for deploying a project with high privacy risk might help you decide whether the costs are worth the business value gained. Review the guidance in the "Understand Your Obligations and Try to Lower Your Risk" section of "Appendix C: SDL Privacy Questionnaire." If your Privacy Impact Rating is P1, schedule a "sanity check" with your organization's privacy expert. This person should be able to guide you through implementation of a high-risk project and might have other ideas to help you reduce your risk.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 7: Stage 2 – Define and Follow Design Best Practices
- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 8: Stage 3 – Product Risk Assessment
- Microsoft Privacy Guidelines for Developing Software Products and Services

STAGE 3: DESIGN PHASE: ESTABLISH AND FOLLOW BEST PRACTICES FOR DESIGN

The best time to influence a project's trustworthy design is early in its lifecycle. Functional specifications may need to describe security features or privacy features that will be directly exposed to users, such as requiring user authentication to access specific data or user consent before use of a high-risk privacy feature. Design specifications should describe how to implement these features, as well as how to implement all functionality as secure features. We define *secure features* as features whose functionality is well engineered with respect to security, such as rigorously validating all data before processing it or cryptographically robust use of cryptographic APIs. It is important to consider security and privacy concerns carefully and early when you design features, and avoid attempts to add security and privacy near the end of a project's development.

Threat modeling (described in "Stage 4: Design Phase: Risk Analysis") is the other critical security activity that must be completed during the design phase.

SECURITY REQUIREMENTS

- Complete a security design review with a security advisor for any project or portion of a project that requires one. Some low-risk components might not require a detailed security design review.
- Satisfy minimal cryptographic design requirements. (See Chapter 20: SDL Minimum Cryptographic Standards, *The Security Development Lifecycle*, pp. 251-258.) Recent advances in cryptographic research means that some older cryptographic techniques are no longer secure. Similarly, existing crypto libraries and APIs can be incorrectly used. This chapter provides guidelines on correct usage and references on when it is necessary to engage cryptographers.
- When developing with managed code, use strong-named assemblies and request minimal permission. When using strong-named assemblies, do not use APTCA (Allow Partially Trusted Caller Attribute) unless the assembly was approved after a security review. Without specific security review and approval, assemblies that use APTCA will generate FxCop errors and will fail to pass a Final Security Review (FSR).
- Be logical and consistent when you make firewall exceptions. Any product or component that requires changes to the host firewall settings must adhere to the requirements that are outlined in "Appendix D: A Policy for Managing Firewall Configurations."

PRIVACY REQUIREMENTS

- If your project has a privacy impact rating of P1, identify a compliant design based on the concepts, scenarios, and rules in the Microsoft Privacy Guidelines for Developing Software Products and Services. Definitions of privacy rankings (P1, P2, P3) can be found in *Stage 2: Cost Analysis* and Chapter 8 of *The Security Development Lifecycle*. You can find additional guidance in "Appendix C: SDL Generic Privacy Questionnaire".

SECURITY RECOMMENDATIONS

- Include in all functional and design specifications a section that describes impacts on security.
- Write a security architecture document that provides a description of a software project that focuses on security. Such a document should complement and reference existing traditional development collateral without replacing it. A security architecture document should contain, at a minimum:
  - Attack surface measurement. After all design specifications are complete, define and document what will be the program's default and maximum attack surfaces. The size of the attack surface indicates the likelihood of a successful attack. Therefore your goal should be to minimize the attack surface. You can find additional background information in the papers Fending Off Future Attacks by Reducing Attack Surface and Measuring Relative Attack Surfaces.
  - Product structure or layering. Highly structured software with well-defined dependencies among components is less likely to be vulnerable than software with less structure. Ideally, software should be structured in a layered hierarchy so that higher components (layers) depend on lower ones. Lower layers should never depend on higher ones. Developing this sort of layered design is difficult and might not be feasible with legacy or pre-existing software. However, teams that develop new software should consider layered and highly structured designs.
- Minimize default attack surface/enable least privilege.
  - All feature specifications should consider whether the features should be enabled by default. If a feature is not used frequently, you should disable it. Consider carefully whether to enable by default those features that are used infrequently.
  - If the program needs to create new user accounts, ensure they have as few permissions as possible for the required function and that they also have strong passwords.

  - Be very aware of access control issues. Always run code with the fewest possible permissions. When code fails, find out why it failed and fix the problem instead of increasing permissions. The more permissions any code has, the greater its exposure to abuse.
- Default installation should be secure. Review functionality and exposed features that are enabled by default and constitute the attack surface carefully for vulnerabilities.
- Defense in depth. Consider a defense-in-depth approach. The most exposed entry points should have multiple protection mechanisms to reduce the likelihood of exploitation of any security vulnerabilities that might exist. If possible, review public sources of information for known vulnerabilities in competitive products, analyze them, and adjust your product's design accordingly.
- If the program is a new release of an existing product, examine past vulnerabilities in previous versions of the product and analyze their root causes. This analysis might uncover additional instances of the same classes of problems.
- Deprecate outdated functionality. If the product is a new release of an existing product, evaluate support for older protocols, file formats, and standards and strongly consider removing them in the new release. Older code written when security awareness was less prevalent almost always contains security vulnerabilities.
- Conduct a security review of all sample source code released with the product and use the same level of scrutiny as for object code released with the product.
- If the product is a new release of an existing product, consider migration of any possible legacy code from unmanaged code to managed code.
- Implement any new code using managed code whenever possible.
- When developing with managed code, take advantage of .NET security features:
  - Refuse unneeded permissions.
  - Request optional permissions.
  - Use CodeAccessPermission Assert and LinkDemand carefully. Use Assert in as small a window as possible.
  - Disable tracing and debugging before deploying ASP.NET applications.
- Watch for ambiguous representation issues. Hackers will try to force code to follow a dangerous path or URL by hiding their intent in escape characters or obscure conventions. Always design code to deal with full canonical representations, rather than acting on externally provided data. The canonical representation of something is the standard, most direct, and least ambiguous way to represent it.
- Remain informed about security issues in the industry. Attacks and threats evolve constantly, and staying current is important. Keep your team informed about new threats and vulnerabilities.
- Ensure that everyone on your team knows about unsafe functions and coding patterns. Maintain a list of your code's vulnerabilities. When you find new vulnerabilities, publish them. Make security everyone's business.
- Be careful with error messages. Sensitive information displayed in an error message can provide an attacker with privileged information, such as a file path on a server or the structure of a query. Such information makes it easier for an attacker to attack any defenses. In general, record detailed failure messages in a secure log, and give the user discreet failure messages.

PRIVACY RECOMMENDATIONS

- If your project has a privacy impact rating of P2, identify a compliant design based on the concepts, scenarios, and rules in the Microsoft Privacy Guidelines for Developing Software Products and Services. Additional guidance can be found in "Appendix C: SDL Generic Privacy Questionnaire".
- Use FxCop to enforce some design guidelines in managed code. Many rules are built in by default.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 8: Stage 3 – Product Risk Assessment
- *Writing Secure Code, Second Edition* (ISBN 9780735617223; ISBN-10 0-7356-1722-8), Appendix C, "A Designer's Security Checklist" (p. 729).
- Fending Off Future Attacks by Reducing Attack Surface, an MSDN article on the process for determining attack surface.
- Measuring Relative Attack Surfaces, a more in-depth research paper.

STAGE 4: DESIGN PHASE: RISK ANALYSIS

During the design phase of development, carefully review security and privacy requirements and expectations to identify security concerns and privacy risks. It is efficient to identify and address these concerns and risks during the design phase

For security concerns, threat modeling is a systematic process that is used to identify threats and vulnerabilities in software. You must complete threat modeling during project design. A team cannot build secure software unless it understands the assets the project is trying to protect, the threats and vulnerabilities introduced by the project, and details of how the project will mitigate those threats.

Important risk analysis considerations include the following:

- **Threats and vulnerabilities** that exist in the project's environment or that result from interaction with other systems. You cannot consider the design phase complete unless you have a threat model or models that include such considerations. Threat models are critical components of the design phase and reference a project's functional and design specifications to describe vulnerabilities and mitigations.
- **Code** that was created by external development groups in either source or object form. It is very important to evaluate carefully any code from sources external to your team. Failure to do so might cause security vulnerabilities about which the project team is unaware.
- **Threat models** that include all legacy code if the project is a new release of an existing program. Such code could have been written before much was known about software security, and therefore could contain vulnerabilities.
- **A Review** of the design of high-risk (P1) privacy projects with a privacy subject matter expert and, if necessary, with appropriate legal counsel conducted as soon as possible in the project. Definitions of privacy rankings (P1, P2, P3) can be found in *Stage 2: Cost Analysis* and Chapter 8 of *The Security Development Lifecycle*.
- **A detailed privacy analysis** to document your project's key privacy aspects. Important issues to consider include:
  - What personal data is collected
  - What is the compelling customer value proposition and business justification
  - What notice and consent experiences are provided
  - What controls are provided to users and enterprises
  - How is unauthorized access to personal information prevented

SECURITY REQUIREMENTS

- Complete threat models for all functionality identified during the cost analysis phase. 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.

- Ensure that all threat models meet minimal threat model quality requirements. All threat models must contain data flow diagrams, assets, vulnerabilities, and mitigation. Threat modeling can be done in a variety of ways using either tools or documentation/specifications to define the approach. For assistance in creating threat models, see "Chapter 9: Stage 4 – Risk Analysis" in *The Security Development Lifecycle* book or consult other guidance listed in the following "Resources" section.
- Have all threat models and referenced mitigations reviewed and approved by at least 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 reviews to ensure the threat models are as comprehensive as possible.
- Threat model data and associated documentation (functional/design specs) have been stored using the document control system used by product team.

PRIVACY REQUIREMENTS

If a project has a privacy impact rating of P1:

- Complete the "Detailed Privacy Analysis" section in "Appendix C: SDL Privacy Questionnaire". The questions will be customized to the behaviors specified in the initial assessment.
- Hold a design review with your privacy subject matter expert.

If your project has a privacy impact rating of P2:

- Complete the "Detailed Privacy Analysis" section in "Appendix C: SDL Privacy Questionnaire". The questions will be customized to the behaviors specified in the initial assessment.
- Hold a design review with your privacy subject matter expert only if one or more of these criteria apply:
    - The privacy subject matter expert requests a design review.
    - You want confirmation that the design is compliant.
    - You wish to request an exception.

If your project has a privacy impact rating of P3, there are no privacy requirements during this phase.

SECURITY RECOMMENDATIONS

- The person who manages the threat modeling process should complete threat modeling training before working on threat models.
- After all specifications and threat models have been completed and approved, the process for making changes to functional or design specifications (known as design change requests, or DCRs) should include an assessment of whether the changes alter existing threats, vulnerabilities, or the effectiveness of mitigations.
- Create an individual work item for each vulnerability listed in the threat model so that your quality assurance team can verify that the mitigation is implemented and functions as designed.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 8: Stage 3 – Product Risk Assessment
- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 9: Stage 4 – Risk Analysis
- Threat Modeling: Uncover Security Design Flaws Using The STRIDE Approach
- Threat Modeling Tool

- Fending Off Future Attacks by Reducing Attack Surface, an MSDN article that describes the process for determining attack surface
- Measuring Relative Attack Surfaces, a more in-depth research paper.

STAGE 5: IMPLEMENTATION PHASE: CREATING DOCUMENTATION AND TOOLS FOR USERS THAT ADDRESS SECURITY AND PRIVACY

Every release of a software program should be secure by design, in its default configuration, and in deployment. However, people use programs differently, and not everyone uses a program in its default configuration. You need to provide users with enough security information so they can make informed decisions about how to deploy a program securely. Because security and usability might conflict, you also need to educate users about the threats that exist as well as the balance between risk and functionality when deciding how to deploy and operate software programs.

It is difficult to discuss specific security documentation needs before development plans and functional specifications stabilize. As soon as the architecture is reasonably stable, the user education team can develop a security documentation plan and schedule. Delivering documentation about how to use a software program securely is just as important as delivering the program itself.

SECURITY RECOMMENDATIONS

- Development management, program management, and user education teams should meet to identify and discuss what information users will need to use the software program securely. Define realistic use and deployment scenarios in functional and design specifications. Consider user needs for documentation and tools.
- User education teams should establish a plan to create user-facing security documentation. This plan should include appropriate schedules and staffing needs. Communicating the security aspects of a program to the user in a clear and concise fashion is as important as insuring that the product code or functionality is free of vulnerabilities.
- For new versions of existing programs, solicit or gather comments about what problems and challenges users faced when securing prior versions.
- Make information about secure configurations available separately or as part of the default product documentation and/or help files. Consider the following issues:
  - The program will follow the best practice of reducing the default attack surface. However, what should users know if they need to activate additional functionality? What risks will they be exposed to?
  - Are there usage scenarios that allow users to lock down or *harden* the program more securely than the default configuration without losing functionality? Inform users about how to configure the program for these situations. Better yet, provide easy-to-use templates that implement such configurations.
  - Inform users about security best practices, such as removing guest accounts and default passwords. External security notes from threat modeling are good sources of information to consider.
  - For programs that use network or Internet communications, describe all communications channels and ports, protocols, and communications configuration options (and their associated security impacts).
  - To support earlier versions of the software and older protocols, it is often necessary to operate less securely. Do not enable insecure protocols in the default configuration. You might still need to deliver them with the release, so inform users about the security implications of older protocols and backward compatibility. Inform users about these trade-offs and how to disable older compatibility modes to achieve the best possible security.
  - Tell users how they can ensure safe use of the program or take advantage of built-in security and privacy features.

PRIVACY RECOMMENDATIONS

- If the program contains privacy controls, create deployment guides for organizations to help them protect their users' privacy (for example, Group Policy controls).
- Create content to help users protect their privacy when using the program (for example, secure your subnet).

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 10: Stage 5 – Creating Security Documents, Tools, and Best Practices for Customers
- Templates contained in the *Windows Server 2003 Security Guide*

STAGE 6: IMPLEMENTATION PHASE: ESTABLISH AND FOLLOW BEST PRACTICES FOR DEVELOPMENT

To detect and remove security issues early in the development cycle, it helps to establish, communicate, and follow effective practices for developing secure code. A number of resources, tools, and processes are available to help you accomplish this goal. Investing time and effort to apply effective practices early will help you eliminate security issues and avoid having to respond to them later in the development cycle, or even after release, which is expensive.

SECURITY REQUIREMENTS

- **Build tools**. Use the currently required (or later) versions of compilers to compile options for the Win32, Win64, WinCE and Macintosh target platforms, as listed in "Appendix E: SDL Required and Recommended Compilers, Tools, and Options for All Platforms".
  - Compile C/C++ code with /GS or approved alternative on other platforms
  - Link C/C++ code with /SAFESEH or approved alternative on other platforms
  - Link C/C++ code with /NXCOMPAT (for more information, refer to "Appendix F: SDL Requirement: No Executable Pages") or approved alternative on other platforms
  - Use MIDL with /robust or approved alternative on other platforms
- **Code analysis tools**. Use the currently required (or later) versions of code analysis tools for either native C and C++ or managed (C#) code that are available for the target platforms, as listed in "Appendix E: Required and Recommended Compilers, Tools, and Options for All Platforms".
- **Banned Application Programming Interfaces (APIs)**. New native C and C++ code must not use banned versions of string buffer handling functions. Based on analysis of previous Microsoft Security Response Center (MSRC) cases, avoiding use of banned APIs is one actionable way to remove many vulnerabilities. For more information, see Security Development Lifecycle (SDL) Banned Function Calls and *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 19: SDL Banned Function Calls, pp. 241-49.
- **No writable shared PE sections**. Sections marked as *shared* in shipping binaries represent a security threat. Use properly secured dynamically created shared memory objects instead. See "Appendix G: SDL Requirement: Shared Section".

PRIVACY REQUIREMENTS

Establish and document development best practices for the development team. Communicate any design changes that affect privacy to your team's privacy lead so that they can document and review any changes.

SECURITY RECOMMENDATIONS

- Comply with minimal Standard Annotation Language (SAL) code annotation recommendations as described in "Appendix H: SDL Standard Annotation Language (SAL) Recommendations for Native Win32 Code". Annotating code helps existing code analysis tools identify implementation issues better and also helps improve the tools. SAL annotated code has additional code analysis requirements, as described in SDL SAL Recommendations.
- All executable programs written using unmanaged code (.EXE) should call the HeapSetInformation interface, as described in "Appendix I: SDL Requirement Heap Manager Fail Fast Setting". Calling this interface will help provide additional defense-in-depth protection against heap–based exploits (Win32 only).
- Review available information resources to adopt appropriate coding techniques and methodologies. For a current and complete list of all development best practice information and resources, see *Writing Secure Code, Second Edition* (ISBN 9780735617223; ISBN-10 0-7356-1722-8).
- Review recommended development tools and adopt appropriate tools, in addition to the tools required by SDL. These tools can be found in *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 21: SDL-Required Tools and Compiler Options.
- Define, document, and communicate to your entire team all best practices and policies based on analysis of all the resources and tools listed in this document.
  - Document all tools that are used, including compiler versions, compile options (for example, /GS), and additional tools used. Also, forecast any anticipated changes in tools. For more information about minimum tool requirements and related policy, review the section "How Are New Recommendations and New Requirements Added to the Security Development Life Cycle Process?" in the "Introduction" section of this document.
  - Create a coding checklist that describes the minimal requirements for any checked-in code. This checklist can include some of the items from *Writing Secure Code, Second Edition* Appendix D, "A Developer's Security Checklist" (p. 731), clean compile warning level requirements (/W3 as minimal, and /W4 clean as ideal), or other desired minimum standards.
  - Establish and document how the team will enforce these practices. Is the team running scripts to check for compliance when code is checked in? How often do you run analysis tools? The development manager is ultimately responsible for establishing, documenting, and validating compliance of development best practices.

DEVELOPER/CODING RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 11: Stage 6 – Secure Coding Policies, Chapter 19: SDL Banned Function Calls
- Compiler Security Checks in Depth
- SAL Annotations

STAGE 7: VERIFICATION PHASE: SECURITY AND PRIVACY TESTING

Security testing addresses two broad areas of concern:

- Confidentiality, integrity, and availability of the software and data processed by the software. This area includes all features and functionality designed to mitigate threats as described in the threat model.

- Freedom from issues that could result in security vulnerabilities. For example, a buffer overrun in code that parses data could be exploited in ways that have security implications.

Begin security testing very soon after the code is written. This testing stage requires one full test pass after the verification stage because potential issues and vulnerabilities might change during development.

Security testing is important to the Security Development Lifecycle. As Michael Howard and David LeBlanc write in *Writing Secure Code, Second Edition*: "The designers and the specifications might outline a secure design, the developers might be diligent and write secure code, but it's the testing process that determines whether the product is secure in the real world."

SECURITY REQUIREMENTS

- *File fuzzing* is a technique that security researchers use to search for security issues. You can find many different fuzzing tools on the Internet. There is also a simple fuzzer on the CD that accompanies the book *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0). Fuzzers can be very general or tuned for particular types of interfaces. File fuzzing requires development teams to make a modest resource investment, but it has uncovered many bugs. You must conduct fuzz testing with "retail" (not debug) builds and must correct all issues as described in the "SDL Privacy Bug Bar and SDL Security Bug Bar (Samples)" appendices.
- If the program exposes remote procedure call (RPC) interfaces, you must use an RPC fuzzing tool to test for problems. You can find RPC fuzzers on the Internet. This requirement applies only to programs that expose RPC interfaces. All fuzz testing must be conducted using "retail" (not debug) builds and must correct all issues as described in the SDL Bug Bar.
- If the project uses ActiveX controls, use an ActiveX fuzzer to test for problems ActiveX controls pose a significant security risk and require fuzz testing. You can find ActiveX fuzzers on the Internet. Conduct all fuzz testing using "retail" (not debug) builds, and correct all issues as described in the "SDL Privacy Bug Bar and SDL Security Bug Bar (Samples)" appendices.
- Satisfy Win32 testing requirements as described in "Appendix J: SDL Requirement: Application Verifier". The Application Verifier is easy to use and identifies issues that are MSRC patch class issues in unmanaged code. AppVerifier requires a modest resource investment and should be used throughout the testing cycle. AppVerifier is not optimized for managed code.
- Define a security bug bar and use it to rate, file, and fix all security vulnerabilities. Vulnerabilities include (in order from most severe to least severe):
  - Elevation of privilege (the ability either to execute arbitrary code or to obtain more privilege than intended)
  - Denial of service
  - Targeted information disclosure (where the attacker can locate and read information from anywhere on the system, including system information, that was not intended or designed to be exposed)
  - Spoofing
  - Tampering (permanent modification of any user data or data used to make trust decisions in a common or default scenario that persists after restarting the operating system or application)

SECURITY RECOMMENDATIONS

- Create and complete security testing plans that address these issues:
  - **Security features and functionality work as specified**. Ensure that all security features and functionality that are designed to mitigate threats perform as expected.
  - **Security features and functionality cannot be circumvented**. If a mitigation can be bypassed, an attacker can try to exploit software weaknesses, rendering security features and functionality useless.

  - **Ensure general software quality in areas that can result in security vulnerabilities**. Validating all data input and parsing code against malformed or unexpected data is a common way attackers try to exploit software. Data fuzzing is a general testing technique that can help prevent such attacks.
- **Penetration testing**. Use the threat models to determine priorities, test, and attack the software as a hacker might. Use existing tools or design new tools if needed.
  - **Hire third-party security firms as appropriate**. Depending on the business goals for your project and availability of resources, consider engaging an external security firm for a security review and/or penetration testing.
- **Develop and use vulnerability regression tests**. If the code has ever had a security vulnerability reported, we strongly suggest that you add regression tests to the test suite for that component to ensure that similar vulnerabilities are not inadvertently re-introduced to the code. Similarly, if there are other products with similar functionality in the market that have suffered publicly reported vulnerabilities, add tests to the test plan to prevent similar vulnerabilities.

PRIVACY RECOMMENDATIONS

- For P1 and P2 projects, include privacy testing in your master test plan. Privacy testing of platform components deployed in organizations should include verification of organizational policy controls that affect privacy (these controls are listed in "Appendix C: SDL Privacy Questionnaire"). Privacy testing for features that transfer data over the Internet should include monitoring network traffic for unexpected network calls.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 12: Stage 7 – Secure Testing Policies
- *How to Break Software: A Practical Guide to Testing* (ISBN 978-0201796193; ISBN-10 0201796198)
- *How to Break Software Security: Effective Techniques for Security Testing* (ISBN 978-0321194336; ISBN-10 0321194330)

STAGE 8: VERIFICATION PHASE: SECURITY PUSH

A security push is a team-wide focus on threat model updates, code review, testing, and thorough documentation review and edit. A security push is not a substitute for a lack of security discipline. Rather, it is an organized effort to uncover changes that might have occurred during development, improve security in any legacy code, and identify and remediate any remaining vulnerabilities. However, it should be noted that it is not possible to build security into software with only a security push.

A security push occurs after a product has entered the verification stage (reached code/feature complete). It usually begins at about the time Beta testing starts. Because the results of the security push might alter the default configuration and behavior of a product, you should perform a final Beta test review after the security push is complete and after all bugs and required changes are resolved.

It is important to note that the goal of a security push is to find bugs, not to fix them. The time to fix bugs is after you complete the security push.

PUSH PREPARATION

A successful push requires planning:

- You should allocate time and resources for the push in your project's schedule, before you begin development. To rush the security push will cause problems or delays during the final security review.

- Your team's security coordinator should determine what resources are required, organize a security push leadership team, and create the needed supporting materials and resources
- The security representative should determine how to communicate security push information to the rest of the team. It is helpful to establish a central intranet location for all information related to the push, including news, schedules, plans, forms and documents, white papers, training schedules, and links. The intranet site should link to internal resources that help the group execute the security push. This site should serve as the primary source of information, answers, and news for employees during the push.
- There must be well-defined criteria to determine when the push is complete.

Your team will need training before the push. At a minimum, this training should help team members understand the intent and logistics of the push itself. Some members might also require updated security training, as well as training in security or analysis techniques that are specific to the software that is undergoing the push. The training should have two components: the push logistics, delivered by a senior member of the team conducting the push, and technical and role-specific security training.

PUSH DURATION

The amount of time, energy, and team-wide focus that a security push requires will differ depending on the status of the code base and the amount of attention the team has given to security earlier in development. A security push will require less time if your team has:

- Rigorously kept all threat models up to date.
- Actively and completely subjected those threat models to penetrations testing.
- Accurately tracked and documented attack surfaces and any changes made to them.
- Completed security code reviews for all high-priority code (see discussion later in this section for details about how priority is assessed).
- Identified and documented development and testing contacts for all code released with the product.
- Rigorously brought all legacy code up to current security standards.
- Validated the security documentation plan.

The duration of a security push is determined by the amount of code that needs to be reviewed for security. Try to conduct security code reviews throughout development, after the code is fairly stable. If you try to condense too many code reviews into too brief a time period, the quality of code reviews will suffer. In general, a security push is measured in weeks, not days. You should aim to complete the push in three weeks and extend the time as necessary.

SECURITY REQUIREMENTS

- **Review and update threat models**. Examine the threat models that were created during the design phase. If circumstances prevented creation of threat models during design phase, you must develop them in the earliest phase of the security push.
- **Review all bugs that affect security against the security bug bar**. Ensure that all security bugs contain the security bug bar rating.

PRIVACY REQUIREMENTS

Review and update the SDL Privacy Questionnaire form (Appendix C to this document) for any material privacy changes that were made during the implementation and verification stages. Material changes include:

- Changing the style of consent

- Substantively changing the language of a notice
- Collecting different data types
- Exhibiting new behavior

SECURITY RECOMMENDATIONS

- **Conduct security code reviews for at-risk components**. Use the following information to help determine which components are most at risk, and use this determination to set priorities for security code review. High-risk items (Pri1) must be reviewed earliest and most in depth. For a minimal checklist for security issues to be aware of during code reviews, see Appendix D: A Developer's Security Checklist in *Writing Secure Code, Second Edition* (p. 731).
- **Identify development and testing owners for everything in the program**. Identify a development owner for each source code file. Identify a quality assurance owner for each binary file. Record this information in a document or spreadsheet and use a document/source tracking system to store it.
- **Prioritize all code before you start the push**. Track priority ratings in a document or spreadsheet that lists the development and quality assurance owners. Subject all code to the same criteria for prioritization, including legacy code. Many security vulnerabilities have come from legacy code that was created before the introduction of security pushes, threat modeling, and the other processes that are included in the Security Development Lifecycle.
- **Ensure that you include and prioritize all sample code shipped with the product**. Consider how users will use the samples. Samples that are expected to be compiled and used with small changes in production environments should be considered Pri1.
- **Re-evaluate the attack surface of the software**. It is important to re-evaluate your team's definition of attack surface during the security push. You should be able to calculate the attack surface based on information described in the design specifications for the software. Measurement of the attack surface will enable you to understand which components have direct exposure to attack and the highest risk of damage if a security breach occurs, Focus effort on areas of highest risk areas, and take appropriate corrective actions. These actions might include:
  - Prolonging the push for especially error-prone components.
  - Deciding not to ship a component until it is corrected.
  - Disabling a component by default.
  - Redesignating a component for future removal from the software (deprecating it).
  - Modifying development practices to make vulnerabilities less likely to be introduced by future modifications or new developments.

  After you evaluate the attack surface, update attack surface documentation as appropriate.
- As time permits, consider code reviews for all Pri2 components.
- Review the security documentation plan. Examine how any changes to the product design during development have affected security documentation. Ensure that the security documentation plan will meet all user needs.
- Focus the entire team on the push. When team members finish reviewing and/or testing their own components, they should help others in the group.

Code priority definitions are provided in the following list:

- Pri1 code is considered the most sensitive from a security standpoint. The following examples of Pri1 code are not necessarily a definitive list:
  - All Internet- or network-facing code
  - Code in the Trusted Computing Base (TCB) (for example, kernel or SYSTEM code)
  - Code running as administrator or Local System

    - Code running as an elevated user (including LocalService and NetworkService)
    - Features with a history of vulnerability, regardless of version
    - Any code that handles secret data, such as encryption keys and passwords
    - Any unverifiable managed code (any code that the standard PEVerify.exe tool reports as not verified)
    - All code supporting functionality exposed on the maximum attack surface
- Pri2 is optionally installed code that runs with user privilege, or code that is installed by default that does not meet the Pri1 criteria.
- Pri3 is rarely used code and setup code. (Setup code that handles secret data, such as encryption keys and passwords, is always considered Pri1 code.)
- Any code or component that has experienced large numbers of security bugs is considered Pri1 code, even if it would otherwise be considered Pri2 or Pri3. Although the definition of large numbers is subjective, it is important to scrutinize carefully the portions of code that contain the most security bugs.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 13: Stage 8 – The Security Push

STAGE 9: PRE-RELEASE PHASE: PUBLIC RELEASE PRIVACY REVIEW

Before any public release (including Alpha and Beta test releases), update the appropriate SDL Generic Privacy Questionnaire for any significant privacy changes that were made during implementation verification. Significant changes include changing the style of consent, substantively changing the language of a notice, collecting different data types, and exhibiting new behavior.

Although privacy requirements must be addressed before any public release of code, security requirements need not be addressed before public release. However, you must complete a final security review before final release.

PRIVACY REQUIREMENTS

- **Review and update the Privacy Companion form**.
    - For a P1 project, your privacy advisor will review your final SDL Privacy Questionnaire (Appendix C to this document), help determine whether a privacy disclosure statement is required, and give final privacy approval for public release.
    - For a P2 project, you need validation by a privacy advisor if any of the following is true:
        - A design review is requested by a privacy advisor.
        - You want confirmation that the design is compliant with privacy standards.
        - You wish to request an exception.
    - For a P3 project, there are no additional privacy requirements.
- **Complete the privacy disclosure**.
    - Draft a privacy disclosure statement as advised by the privacy advisor. If your privacy advisor indicates that a privacy disclosure is waived or covered, you do not need to meet this requirement.
    - Work with your privacy advisor and legal representatives to create an approved privacy disclosure.
    - Post the privacy disclosure to the appropriate Web site before each public release.

PRIVACY RECOMMENDATIONS

- Create talking points as suggested by the privacy advisor to use after release to respond to any potential privacy issues.
- Review deployment guidance for enterprise programs to verify that privacy controls that affect functionality are documented. Conduct a legal review of the deployment guide.
- Create "quick text" for your support team that addresses likely user questions, and generally foster strong and frequent communication between your development and support teams.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 14: Stage 9 – The Final Security Review

STAGE 10: RELEASE PHASE: RESPONSE PLANNING

Any software can be released with unknown security issues or privacy issues, despite best efforts and intentions. Even programs with no known vulnerabilities at the time of release can be subject to new threats that emerge and that might require action. Similarly, privacy advocates might raise privacy concerns after release. You must prepare before release to respond to potential security and privacy incidents. With proper planning, you should be able to address many of the incidents that could occur in the course of normal business operations.

Your team must be prepared for a *zero-day exploit* of a vulnerability—one for which a security update does not exist. Your team must also be prepared to respond to a software security emergency. If you create an emergency response plan before release, you will save time, money and frustration when an emergency response is required for either security or privacy reasons.

SECURITY REQUIREMENTS

- The project team must provide contact information for people who respond to security incidents. Typically, such responses are handled differently for products and services.
  - Provide information about which existing sustained engineering (SE) team has agreed to be responsible for security incident response for the project. If the product does not have an identified SE team, they must provide an emergency response plan (ERP) and provide it to the incident response team. This plan must include contact information for 3-5 engineering resources, 3-5 marketing resources, and 1-2 management resources who will be the first points of contact when you need to mobilize your team for a response effort. Someone must be available 24 hours a day, 7 days a week – and contacts must understand their roles and responsibilities and be able to execute on them when necessary.
  - Identify someone who will be responsible for security servicing. All code developed outside the project team (third-party components) must be listed by filename, version, and source (where it came from).
  - You must have an effective security response process for servicing code that has been inherited or reused from other teams. If that code has a vulnerability, the releasing team may have to release a security update even though it did not develop the code.
  - You must also have an effective security response process for servicing code that has been licensed from third parties in either object or source form. For licensed code, you also need to consider contractual requirements regarding which party has rights and obligations to make modifications, associated service level agreements, and redistribution rights for any security modifications.

- Create a documented sustaining model that addresses the need to release immediate patches in response to security vulnerabilities and does not depend entirely on infrequent service packs.
- Develop a consistent and comprehensible policy for security response for components that are released outside of the regular product release schedule (out of band) but that can be used to update or enhance the software after release. For example, Windows must plan a response to security vulnerabilities in a component such as DirectX that ships as part of the operating system, but that might also be updated independently of the operating system, either directly by the user or by the installation of other products or components.

PRIVACY REQUIREMENTS

- For P1 and P2 projects, identify the person who will be responsible for responding to all privacy incidents that may occur. Add this person's e-mail address to the "Incident Response" section of the SDL Privacy Questionnaire (Appendix C to this document). If this person changes positions or leaves the team, identify a new contact and update all SDL Generic Privacy Questionnaire forms for which that person was listed as the privacy incident response lead.
- Identify additional development and quality assurance resources on the project team to work on privacy incident response issues. The privacy incident response lead is responsible for defining these resources in the "Incident Response" section of the SDL Generic Privacy Questionnaire.
- After release, if a privacy incident occurs you must be prepared to follow the SDL Privacy Escalation Response Framework (Appendix K to this document), which might include risk assessment, detailed diagnosis, short-term and long-term action planning, and implementation of action plans. Your response might include creating a patch, replying to media inquiries, and reaching out to influential external contacts.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 15: Stage 10 – Security Response Planning
- Microsoft Support Lifecycle home page
- "Appendix K: SDL Privacy Escalation Response Framework (Sample)"

STAGE 11: RELEASE PHASE: FINAL SECURITY REVIEW AND PRIVACY REVIEW

As the end of your software development project approaches, you need to be sure that the software is secure enough to ship. The final security review (FSR) will help determine this. The security team assigned to the project should perform the FSR with help from the product team to ensure that the software complies with all SDL requirements and any additional security requirements identified by the security team (such as penetration testing or additional fuzz testing).

A final security review can last anywhere from a few days to six weeks, depending on the number of issues and the team's ability to make necessary changes.

It is important to schedule the FSR carefully—that is, you need to allow enough time to address any serious issues that might be found during the review. You also need to allow enough time for a thorough analysis; insufficient time could cause you to make significant changes after the FSR is completed.

THE FSR PROCESS

- Define a due date for all project information that is required to start the FSR. To minimize the likelihood of unexpected delays, plan to conduct an FSR 4-6 weeks before Release to Manufacture (RTM) or Release to Web (RTW). Your team might need to revalidate specific decisions or change code to fix security issues. The team must understand that additional security work will need to be performed during the FSR.

- The FSR cannot begin until you have completed the reviews of the security milestones that were required during development. Milestones include in-depth bug reviews, threat model reviews, and running all SDL-mandated tools.
    - Reconvene the development and security leadership teams to review and respond to the questions posed during the FSR process.
    - Review threat models. The security advisor should review the threat models to ensure that all known threats and vulnerabilities are identified and mitigated. Have complete and up-to-date threat models at the time of the review.
    - Review security bugs that were deferred or rejected for the current release. The bug review should ensure that a consistent, minimum security standard was adhered to throughout the development cycle. Teams should already have reviewed all security bugs against the criteria that were established for release. If the release does not have a defined security bug bar, your team can use the standard SDL Security Bug Bar.
    - Validate results of all security tools. You should have run these tools before the FSR, but a security advisor might recommend that you also run other tools. If tool results are inaccurate or unacceptable, you might need to rerun some tools.
    - Ensure that you have done all you can to remove bugs that meet your organization's severity criteria, so that there are no known vulnerabilities. Ultimately, the goal of SDL is to remove security vulnerabilities from products and services. No software release can pass an FSR with known vulnerabilities that would be considered as Sev 1, Sev 2, or Sev 3.
- Submit exception requests to a security advisor for review. If your team cannot meet a specific SDL requirement, you must request an exception. Typically, such a request is made well in advance of the FSR. A security advisor will review these requests and, if the overall security risk is tolerable, might choose to grant the exception. If the security risk is not acceptable, the security advisor will deny the exception request. It is best to address all exception requests as soon as possible in the development phase of the project.

POSSIBLE FSR OUTCOMES

Possible outcomes of an FSR include:

- **Passed FSR**. If all issues identified during the FSR are corrected before RTM/RTW, the security advisor should certify that the project has successfully met all SDL requirements.
- **Passed FSR (with exceptions)**. If all issues identified during the FSR are corrected before RTM/RTW *OR* the security advisor and team can reach an acceptable compromise about any SDL requirements that the project team was unable to resolve. The security advisor should identify the exceptions and certify that all other aspects of the project have successfully met all SDL requirements.
    - All exceptions and security issues not addressed in the current release should be logged, and then addressed and corrected in the next release.
- **FSR 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 correct whatever SDL requirements that they cannot correct or escalate to higher management for a decision.
    - Escalations occur when the security advisor determines that a team cannot meet the defined requirements or is in violation of an SDL requirement. Typically, the team has a business justification that prevents them from being compliant with the requirement. In such instances, the security advisor and the team should work together to compose a consolidated escalation report that outlines the issue—including a description of the security or privacy

risk and the rationale behind the escalation. This information is typically provided to the business unit executive and the executive with corporate responsibility for security and privacy, to aid decision making.

- If a team fails to follow proper FSR procedures - either by an error of omission or by willful neglect - the result will be an immediate FSR failure. Examples include:
    - Errors of omission, such as failure to properly document all required information
    - Specious claims and willful neglect, including
        - Claims of "Not subject to SDL" contrary to evidence
        - Claims of "FSR pass" contrary to evidence, and software RTM/RTW without the appropriate signoff

  Such an incident can result in very serious consequences subsequently, and is always immediately escalated to the project team executive staff and the executive in charge of security and privacy.

SECURITY REQUIREMENTS

- The project team must provide all required information before the scheduled FSR start date. Failure to do so will delay completion of the FSR. If the schedule slips significantly before the FSR begins, contact the assigned security advisor to reschedule.
- After the FSR is finished, the security advisor will either sign off on the project as-is or provide a list of required changes.

PRIVACY REQUIREMENTS

- Repeat the privacy review for any open issues that were identified in the pre-release privacy review, or for material changes made to the product after the pre-release privacy review. Material changes include modifying the style of consent, substantively revising the language of a notice, collecting different data types, or exhibiting new behavior. If no material changes were made, no additional reviews or approvals are required.
- After the privacy review is finished, your privacy advisor will either sign off on the product as-is or will provide a list of required changes.

SECURITY RECOMMENDATIONS

- Ensure the product team is constantly evaluating the severity of security bugs against the standard that is used during the security push and FSR. Otherwise, a large number of security bugs might be reactivated during the FSR.

RESOURCES

- *The Security Development Lifecycle* (ISBN 9780735622142; ISBN-10 0-7356-2214-0), Chapter 16: Stage 11 – Product Release

STAGE 12: RELEASE PHASE: RTM/RTW

Software release to manufacturing (RTM) or to the Web (RTW) is conditional to completion of the Security Development Lifecycle process as defined in this document. The security advisor assigned to the release must certify that your team has satisfied security requirements. Similarly, for all products that have at least one component with a privacy impact rating of P1, your privacy advisor must certify that your team has satisfied the privacy requirements before the software can be shipped.

SECURITY REQUIREMENTS

- To facilitate the debugging of security vulnerability reports and to help tools teams research cases in which automated tools failed to identify security vulnerabilities, all product teams must submit symbols for all publicly released products as part of the release process. This requirement is needed only for RTM/RTW binaries and any post-release binaries that are publicly released to customers (such as Service Packs, Updates, and so on).
- Design and implement a sign-off process to ensure security and other policy compliance before you ship. This process should include explicit acknowledgement that the product successfully passed the FSR and was approved for release.

PRIVACY REQUIREMENTS

- Design and implement a sign-off process to ensure privacy and other policy compliance before you ship. This process should include explicit acknowledgement that the product successfully passed the FSR and was approved for release.

RESOURCES

N/A

STAGE 13: POST-RELEASE PHASE: RESPONSE EXECUTION

After a software program is released, the product development team must be available to respond to any possible security vulnerabilities or privacy issues that warrant a response. In addition, develop a response plan that includes preparations for potential post-release issues.

RESOURCES

- "Appendix K: SDL Privacy Escalation Response Framework (Sample)”

APPENDIX A: PRIVACY AT A GLANCE

This sample document provides basic criteria to consider when building privacy into software releases. It is not exhaustive and should not be treated as such. For more comprehensive guidance, see Privacy Guidelines for Developing Software Products and Services.

TEN THINGS YOU MUST DO TO PROTECT PRIVACY

- **Collect user data only if you have a compelling business and customer value proposition**. Collect data only if you can clearly explain the net benefit to the customer. If you are hesitant to tell customers what you plan to do, then don’t collect their data.
- **Collect the smallest amount of data for the shortest period of time**. Collect personal data only if you absolutely must, and delete it as soon as possible. If there exists a need to retain personal data, ensure that there is business justification for the added cost and risk. Do not collect data for undefined future use.
- **Collect the least sensitive form of data**. If you must collect data, collect it anonymously if possible. Collect personal data only if you are absolutely certain you need it. If you must include an ID, use one that has a short life span (for example, lasting a single session). Use less sensitive forms of data (for example, telephone area code rather than full phone number). Whenever possible, aggregate personal data from many individuals.
- **Provide a prominent notice and obtain explicit consent before transferring personal data from the customer's computer**. Before you transfer any personal data, you must tell the customer what data will be transferred, how it will be used, and who will have access to it. Important aspects of the transfer must be visible to the customer in the user interface.
- **Prevent unauthorized access to personal data**. If you store or transfer personal data you must help protect it from unauthorized access, including blocking access to other users on the same system, using technologies that help protect data sent over the Internet, and limiting access to stored data.
- **Get parental consent before collecting and transferring a child's personal data**. Special rules for interacting with children apply any time you know the user is a child (because you know the child’s age) or when the content is targeted at or attractive to a child.
- **Provide administrators with a way to prevent transfers**. In an organization, the administrator must have the authority to say whether any data is transferred outside the organization's firewall. You must identify or provide a mechanism that allows the administrator to suppress such transfers. This control must supersede any user preferences.
- **Honor the terms that were in place when the data was originally collected**. If your team decides to use data, its use must be subject to the disclosure terms that were presented to the customer when it was collected.
- **Provide customers access to their stored personal data**. Customers have a right to inspect the personal data you collect from them and correct it if it is inaccurate—especially contact information and preferences. You also need to ensure that the customer is authenticated before he or she is allowed to inspect or change the information.
- **Respond promptly to user questions about privacy**. Inevitably, some users will have questions about your practices. It is essential that you respond quickly to such concerns. Unanswered questions cause a loss of trust. Be sure a member of your staff is ready to respond whenever a user asks about a privacy issue.

APPENDIX B: SECURITY DEFINITIONS FOR VULNERABILITY WORK ITEM TRACKING

It is critical for project teams to specify and maintain a work item tracking system – that allows for creation, triage, assignment, tracking, remediation and reporting of software vulnerabilities. Optimally, work item tracking should also include the ability to track security and privacy issues by cause and effect of the security bugs. The work item tracking system should have access controls in place to ensure that changes to information in the system (whether malicious or accidental) can be tracked appropriately.

Ensure that the vulnerability/work item tracking system used includes fields with the following values (at a minimum):

SECURITY BUG CAUSE

The following fields describe causes of vulnerabilities:

- **Not a Security Bug**. This field is self-explanatory.
- **Buffer Overflow/Underflow**. A failure to check or to limit input data buffer sizes before data is manipulated or processed.
- **Arithmetic Error**. A failure to check bounds conditions for integer math, in which results of calculations might overflow or underflow data type. An example is integer overflow.
- **SQL/Script Injection**. Allows attackers to alter intended behavior by altering script.
- **Directory Traversal**. Allows attackers access to navigate host directory structure.
- **Race Condition**. A security vulnerability caused by code timing or synchronization issues.
- **Cross-Site Scripting**. This cause involves Web site weaknesses that allow attackers to have inappropriate access to information or resources. Although a subset of script injection, it is listed separately.
- **Cryptographic Weakness**. Insufficient or incorrect use of cryptography to protect data.
- **Weak Authentication**. Insufficient checks or tests to validate that the user or process is who or what it claims to be.
- **Weak Authorization/Inappropriate permission or ACL**. Access to resources or data for an authenticated user that are not appropriate for users of that type. For example, allowing anonymous or guest users access to sensitive information.
- **Ineffective Secret Hiding**. Insufficient or incorrect protection of cryptographic keys or passwords. For example, storing passwords in plain text in registry or not zeroing out password buffers.
- **Unlimited Resource Consumption (DoS)**. A failure to check or limit resource allocations that might allow an attacker to deny service by depleting available resources.
- **Incorrect/No Error Messages**. Insufficient or incorrect reporting of error checking.
- **Incorrect/No Pathname Canonicalization**. An incorrect trust decision based on a resource name, or allowing access to a resource because an attacker bypassed location or name restrictions.
- **Other**. None of the above.

SECURITY BUG EFFECT

The following definitions are from *Writing Secure Code, Second Edition*.

- **Not a Security Bug**. This field is self-explanatory.
- **Spoofing**. Spoofing threats allow an attacker to pose as another user, allow a rogue server to pose as a valid server or rogue code to pose as valid code

- **Tampering**. Data tampering means malicious modification of data.
- **Repudiation**. Repudiation threats are associated with users who deny having performed an action without other parties having any way to prove otherwise: for example, a user with malicious intent performs an illegal operation on a computer that is unable to trace the prohibited operation.
- **Information Disclosure**. Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it. For example, such a threat might be a user's ability to read a file to which they did not have access, or an intruder's ability to read data in transit between two computers.
- **Denial of Service**. Denial of service (DoS) attacks deny or restrict services to valid users.
- **Elevation of Privilege**. In this type of threat, a user increases their permissions level and therefore can perform actions they should not be allowed to perform. Any unprivileged user who gains unauthorized access might have sufficient access to compromise or even destroy the system.
- **Attack Surface Reduction**. This type of threat is not a security bug in the same sense as the other items listed. However, when you attempt to reduce the attack surface, it is valuable to track bugs that describe services and functionality that affect the attack surface. It is important to identify attack surface, even though interfaces that are exposed on the attack surface are technically not vulnerabilities. Such bugs are assigned the *Attack Surface Reduction* designation.

APPENDIX C: SDL PRIVACY QUESTIONNAIRE

This sample document provides some criteria to consider when you build a privacy questionnaire. It is not an exhaustive list, and should not be treated as such.

INTRODUCTION

The following questions are designed to help you complete the privacy aspects of the Security Development Lifecycle (SDL). You will complete some sections, such as the initial assessment and a detailed analysis, on your own. You should complete other sections, such as the privacy review, together with your privacy advisor.

IDENTIFY YOUR PROJECT AND KEY PRIVACY CONTACTS

- What is the name of your project?

  ____

- When will the public first have access to this project?

  ____

- Who on your team is responsible for privacy?

  ____

INITIAL ASSESSMENT

The initial assessment is a quick way to determine your *privacy impact rating* and estimate the work required to be compliant. The rating (P1, P2, or P3) represents the degree of risk your software presents from a privacy perspective. You need to complete only the steps that apply to your rating. For more detail, see the main Microsoft Security Development Lifecycle document.

DETERMINE YOUR PRIVACY IMPACT RATING

Check all behaviors that apply to your software. If your software does not exhibit any of the behaviors, check "None of the above." For more information, see the Privacy Guidelines for Developing Software Products and Services.

___ Store personally identifiable information (PII) on the user's computer or transfer it from the user's computer (P1)

___ Provide an experience that targets children or is attractive to children (P1)

___ Continuously monitor the user (P1)

___ Install new software or change file type associations, home page, or search page (P1)

___ Transfer anonymous data (P2)

___ None of the above (P3)

UNDERSTAND YOUR OBLIGATIONS AND TRY TO LOWER YOUR RISK (FOR P1 AND P2 SCENARIOS)

Before you invest time in a design or implementation, get a feel for the work it will take and investigate ways to lower your overall privacy risk. Higher risk translates to higher development and support cost. For more information, see the Privacy Guidelines for Developing Software Products and Services.

IDENTIFY A COMPLIANT DESIGN

For more information, see the Privacy Guidelines for Developing Software Products and Services.

PERFORM A DETAILED PRIVACY ANALYSIS FOR P1 SCENARIOS

Before your privacy design review, analyze your threat model to identify any PII that you store or transfer. Summarize the privacy aspects of your software in a detailed analysis.

- Describe the PII you store or data you transfer:

  ____

  ____

- Describe your compelling customer value proposition and business justification:

  ____

  ____

- Describe any software you install or changes you make to file types, home page, or search page:

  ____

  ____

- Describe your notice and consent experiences:

  ____

  ____

- Describe how users will access your public disclosure:

  ____

  ____

- Describe how organizations can control your feature:

  ____

  ____

- Describe how users can control your feature:

  ____

  ____

- Describe how you will prevent unauthorized access to PII:

  ____

  ____

CONDUCT A DESIGN REVIEW WITH YOUR PRIVACY ADVISOR

To avoid costly mistakes, projects with an impact rating of P1 must hold a design review with a privacy advisor before investing heavily in implementation.

TEST YOUR PRIVACY EXPERIENCE

Verify that your software complies with privacy requirements. For more information about privacy criteria, see the Privacy Guidelines for Developing Software Products and Services.

CREATE A DRAFT PRIVACY DISCLOSURE

Work with your privacy advisor to write and post a privacy disclosure.

DESIGNATE YOUR PRIVACY INCIDENT RESPONSE CONTACT

If your software is involved in a privacy incident, your team must be prepared to follow the Privacy Escalation Response Framework (Appendix M in this guide).

Who on your team is the primary contact for Privacy Incident Response?

____

OBTAIN APPROVAL FROM YOUR PRIVACY ADVISOR

Before you ship your project externally, you must obtain approval from your privacy advisor.

APPENDIX D: A POLICY FOR MANAGING FIREWALL CONFIGURATIONS

A firewall is a key part of any organization's protection strategy. You should have a consistent policy to manage the settings of your organization's firewall to ensure that users are not exposed to unknown or unnecessary risk from programs that receive unsolicited data over the network.

At least 80% of a program's users require an open port in the firewall and the code that listens on the port must comply with certain quality requirements. Specific distinctions are made between two types of open ports: *port exceptions* and *program exceptions* (see the "Port Exceptions" and "Program Exceptions" sections later in this appendix for more specific information). When an open port is needed, requirements exist for obtaining users' consent, although there is a limited notion of implicit consent. Also, you must perform fuzz testing if your program needs to open a port in the firewall.

FIREWALL POLICY STATEMENTS

This section provides detailed information about various firewall policy components.

SCOPE

This firewall policy applies to software that can receive TCP/IP traffic from across the Internet, such as:

- Software shipped as part of Microsoft® Windows® XP Service Pack 2 (SP2) and later, Windows Vista™, Windows Server® 2003 SP1 and later.
- Layered products (such as Microsoft Office and Microsoft SQL Server™) installed on the referenced Windows platforms.

Specifically excluded from this policy are

- Windows Starter Edition
- Software for MacOS
- Microsoft Windows Mobile® and software for Windows Mobile
- Microsoft Internet Security and Acceleration (ISA) Server
- Windows Live OneCare™
- Microsoft XBox®
- InfraRed Data Association (IrDA), Bluetooth, USB, FireWire, and other personal area and local area networking technologies (specifically, non-TCP/IP networking technologies)

This policy affects unsolicited data from remote computers by default. It addresses firewall settings and management, as well as any mechanisms that bypass the firewall (such as IPsec or additional Windows Filtering Platform filters). You may not bypass this policy by installing IPsec policies or by making other configuration changes if changing the firewall to achieve the same goals violates the policy.

TERMINOLOGY

The following definitions apply to some basic terms in this policy:

- **Exception**. A setting that, when enabled, allows unsolicited traffic through the firewall. In Windows XP SP2, there are two types of exceptions: port exceptions, which allow unsolicited traffic on the specified port, and program exceptions, which allow a specific program to receive unsolicited traffic, regardless of port. An exception setting may be enabled, which means that traffic is allowed to bypass the firewall, or disabled, which has no effect on the firewall. The same concepts apply to a later Windows firewall or to a third-party firewall.

- **Private Network/Address**. A private address is an IP address that, when correctly configured, would not appear on the public Internet. For Internet Protocol version 4 (IPv4), this range includes the addresses defined by RFC1918, RFC3927, and the private-use addresses defined in RFC3330. For Internet Protocol version 6 (IPv6), the situation is less clear. RFC 3518 defines link-local and site-local IPv6 address ranges, but RFC 3879 deprecates this scheme. Until this situation is resolved, all IPv6 addresses should be considered non-private.
- **Public Network/Address**. A public address is an IP address or address space that is accessible from the public Internet.

BASICS

The following actions are never permitted:

- Disabling the firewall. The firewall may be disabled only by explicit user action.
- Any service or feature that enables an exception automatically when the service starts.
- Automatically enabling an exception would effectively nullify the firewall for that port.
- Programs, features, and services that are not designed specifically as firewall management utilities should leave the firewall alone, except at setup time or unless the user has explicitly initiated some action.
- Any service or feature that allows a port to be opened or an exception to be enabled by a user without administrative privileges. A user must act as an administrator to change the settings of the firewall, and no service or feature should bypass this restriction.
- Silent activation or enabling any feature that permits other programs to receive unsolicited traffic. For example, the RemoteAdmin feature permits other RPC–based programs to receive unsolicited traffic. In such cases, the administrator must obtain user consent before activating such functionality.
- Configuring an external device (for example, a NAT gateway) without user consent.
- Interference with Post Setup Security Update or similar functionality designed to ensure that the system is up to date before it accepts incoming traffic.

The following actions are always permitted:

- Defining an exception and leaving it disabled for the user's convenience in choosing to enable the exception later.
- Enabling an exception scoped exclusively to 127.0.0.1 or equivalent.

The following action is always required:

- Logging changes to the firewall settings in the system security log.

PORT EXCEPTIONS

Port exceptions are permitted only if empirical data provided for the final security review shows that at least 80% of users of the program will use a feature that requires the port within the next year, or in one of the following circumstances:

- The user has provided explicit *informed consent*. In this context, informed consent requires either that the default answer for a dialog results in the port remaining closed, or that data shows that 80% of customers will answer the dialog correctly.
- *Implicit consent* can be inferred in certain limited cases for home and organizational networking scenarios. These scenarios include:

  Home networking. Home networking implicit consent is provided when all of the following conditions are true:

  - The product is marketed primarily to home users.
  - The program can detect the presence of other local services, such as Xbox services, media devices, or other services intended for home networking scenarios.

- The port is closed by default any time the interface has an IP address in public IP space.
- The computer is not joined to a domain.
- The port is configured for the local subnet only by default.
- Any network address translation (NAT) traversal technology is in use.
- The user is informed and acknowledges the open port(s) when they are opened the first time. Note that this condition does not require consent. Merely informing the user and having the user click **OK** is sufficient in this case.

Organizational networking. You can consider that implicit consent was provided when all of the following are true:

- The product is marketed primarily to enterprise and/or small and medium-size businesses.
- The product is intended to be used primarily as a server.
- The user is informed about and acknowledges the open port.
- The port can be controlled and managed through Group Policy.

Implicit consent is never provided for the first computer on a home network. Implicit consent in the home scenario is intended primarily to enable certain peer-to-peer scenarios. In this case, the expectation is that the user will have explicitly enabled some home networking functionality on the first computer on the network. Other computers that detect the presence of those features on the local network might implicitly enable those features as well, subject to the other requirements as listed.

Port exceptions apply to all interfaces on a system that can be addressed remotely (either a wired interface or a wireless interface). IrDa and other interfaces that require physical proximity are not addressed by this policy at this time.

PROGRAM EXCEPTIONS

Program exceptions are permitted only if:

- The program does not run as a service.
- The program listens on the network only when the user is using the functionality that listens (including programs that start when the user logs in).
- Users can interact with a GUI to prevent the program from starting automatically.

QUALITY BAR

For both program exceptions and port exceptions, the programs, applications, services, or other components that wish to receive unsolicited traffic must accomplish the following tasks:

- Produce an independent threat model for the service that identifies each entry point explicitly, including services that are multiplexed behind a common port.
- Fuzz test each entry point for 48 hours without service failure or memory leaks.
- Establish a fuzz testing plan.

LEAST PRIVILEGE

Ports that are open by default (whether through a port exception or a program exception) must adhere to the principle of least privilege. Specifically, this requirement means that they must:

- Scope the exception to the local subnet (or more restrictive when practical).

- Limit the privileges of the program that uses the port to Network Service (or more restrictive when practical). When not practical, the threat model should explicitly state why.

If services must run with privileges greater than Network Service, we urge you to split the service into privileged and non-privileged components so that only the code that requires greater privileges receives them and other code is addressed through some inter-process communication (IPC) mechanism. Some background information is available in the paper "Privtrans: Automatically Partitioning Programs for Privilege Separation."

CONSIDERATION OF THIRD-PARTY FIREWALLS

Third-party firewalls are common. Any program that wishes to make a program or port exception must:

- Function gracefully if Windows Firewall is disabled or not present.
- Provide a way for the user to discover which ports must be open in order for the program to work.

USER INTERFACE

Although this firewall policy addresses user interface issues, nothing in the policy should be interpreted to specify a particular user interface. For example, the sentence "The user is informed about and acknowledges the open port" does not imply that there must be a dialog that tells the user port 123 has been opened. The user must only be informed of the change explicitly, and the details must be available for users who want to know them.

RISKS AND ISSUES THAT CAN AFFECT THIS FIREWALL POLICY

The following factors might change this policy:

- **NAT traversal**. The widespread adoption of NAT traversal might invalidate the assumption that permits implicit consent in home networking scenarios. Widespread adoption of NAT traversal technologies might make implicit consent in the home networking scenario an unreasonable risk.
- **Network Location Aware (NLA)-informed profiles** (profiles that can be configured on a per-location basis). When NLA-informed profiles become available, this policy will be modified appropriately. The policy will likely include a requirement that ports remain closed on a per-profile basis.
- **Resolution of what constitutes private IP space for IPv6 addresses**. Resolution of this issue might permit some IPv6 addresses to be treated as private.

This policy fails to protect against some key risks. These risks include:

- **The "Open Network" scenario**. For mobile systems, this policy would permit a situation in which ports are inappropriately enabled on "foreign" networks (for example, if a user configures a system on their home network and then visits a coffee shop). If NLA-informed profiles are available, this risk is substantially diminished.
- Inappropriate opening of Microsoft Windows Media® Center portable systems. This condition might occur when a computer is used in typical business scenarios.

APPENDIX E: REQUIRED AND RECOMMENDED COMPILERS, TOOLS, AND OPTIONS FOR ALL PLATFORMS

WIN32 REQUIREMENTS: UNMANAGED CODE

Compiler/tool	Minimum required version and switches/options	Optimal/recommended version and switches/options	Comments
C/C++ Compiler	Microsoft® Visual Studio® .NET 2005		
cl.exe	Version 14.0.50727.42 Use /GS	Use /GS	
Link.exe	Version 8.0.50727.42 Use /SAFESEH Use /NXCOMPAT and don't use /NXCOMPAT:NO. See "Appendix F: SDL Requirement: No Executable Pages" for more information.	Use /SAFESEH Use /functionpadmin:5 Use /DYNAMICBASE	Visual Studio 2005 SP1 is needed for /DYNAMICBASE
MIDL.exe	Version 6.0.366.1 Use /robust	Use /robust	
Source code analysis	Visual Studio 2005 Code Analysis Options ("/analyze") For Visual Studio 2005 code analysis, all warning IDs from the following list must be fixed: 4532 6029 6053 6057 6059 6063 6067 6200 6201 6202 6203 6204 6248 6259 6260 6268 6276 6277 6281 6282 6287 6288 6289 6290 6291 6296 6298 6299 6305 6306 6308 6334 6383	Visual Studio 2005 Code Analysis Options ("/analyze"). For Visual Studio 2005 code analysis, all warning IDs from the following list must be fixed: 4532 6029 6053 6057 6059 6063 6067 6200 6201 6202 6203 6204 6248 6259 6260 6268 6276 6277 6281 6282 6287 6288 6289 6290 6291 6296 6298 6299 6305 6306 6308 6334 6383 Standard Annotation Language (SAL): Code annotated with SAL should correct additional warnings in addition to those listed above. See "Appendix H: SDL Standard Annotation Language (SAL) Recommendations for Native Win32 Code" for more information. The warnings are summarized as follows: **SAL Compliance** Visual Studio 2005: 26020 - 26023	Visual Studio 2005 Team Edition contains a publicly available version that is branded as "C/C++ Code Analysis".

		/analyze Visual Studio 2005: 6029; 6053; 6057; 6059; 6063; 6067; 6201-6202; 6248; 6260; 6276; 6277; 6305	
Protecting against Heap Corruption	n/a	All executable programs written using unmanaged code (.EXE) must call the HeapSetInformation interface. See "Appendix I: SDL Requirement: Heap Manager Fail Fast Setting" for more information.	

WIN32 REQUIREMENTS: MANAGED CODE

Compiler/tool	**Minimum required version and switches/options**	**Optimal/recommended version and switches/options**	**Comments**
C# Compiler	Visual Studio 2005		
csc.exe	Version 8.0.50727.42		
.NET Framework	Version 2.0.50727		
FxCop	Version 1.32	Most recent version	

WIN32 REQUIREMENTS: TESTING TOOLS

Tool	**Minimum required version and switches/options**	**Optimal/recommended version and switches/options**	**Comments**
AppVerifier	Most recent version Run tests as described in "Appendix J: SDL Requirement: Application Verifier".	Most recent version	**Note**: Appverifier is targeted at unmanaged code and is not optimized for managed code.

WIN64 REQUIREMENTS (IA64 AND AMD64): UNMANAGED CODE

Compiler/tool	**Minimum required version and switches/options**	**Optimal/recommended version and switches/options**	**Comments**
C/C++ Compiler	Visual Studio 2005		
cl.exe	Version 14.0.50727.42		
Link.exe	Version 8.0.50727.42 Use of /SAFESEH does not apply to Win64 platforms. Use /NXCOMPAT and do not use	AMD64 only: Use /functionpadmin:6 Use of /SAFESEH does not apply to Win64 platforms. Use /DYNAMICBASE	Visual Studio 2005 SP1 is needed for /DYNAMICBASE

	/NXCOMPAT:NO. See "Appendix F: SDL Requirement: No Executable Pages" for more information.		
MIDL.exe	Version 6.0.366.1 Use /robust	Use /robust	
Protecting against Heap Corruption	n/a	All executable programs written using unmanaged code (.EXE) must call the HeapSetInformation interface. See "Appendix I: SDL Requirement: Heap Manager Fail Fast Setting" for more information.	

WIN64 REQUIREMENTS (IA64 AND AMD64): MANAGED CODE

Compiler/tool	Minimum required version and switches/options	Optimal/recommended version and switches/options	Comments
C# Compiler	Visual Studio 2005		
csc.exe	Version 8.0.50727.42		
.NET Framework	Version 2.0.50727		
FxCop	Most recent version	Most recent version	

WIN64 REQUIREMENTS (IA64 AND AMD64): TESTING TOOLS

Tool	Minimum required version and switches/options	Optimal/recommended version and switches/options	Comments
AppVerifier	Most recent version Run tests as described in "Appendix J: SDL Requirement: Application Verifier."	Most recent version	**Note**: Appverifier is targeted at unmanaged code and is not optimized for managed code.

WINCE REQUIREMENTS: UNMANAGED CODE

Compiler/tool	Minimum required version and switches/options	Optimal/recommended version and switches/options	Comments
C/C++ Compiler	Visual Studio 2005		
cl.exe	Version 14.0.50727.42 Use –GS (see comments)	Use –GS (see comments)	The –GS flag has a modest impact on code size, which can be of interest on WinCE platforms. Minimally, –GS must

			be used on all Internet-facing code. Ideally, –GS should be used on all code.
Link.exe	Version 8.0.50727.42 Use of /SAFESEH only applies to x86 with WinCE platforms. Use of /NXCOMPAT does not apply to WinCE.	Use of /SAFESEH only applies to x86 on WinCE platforms. Use of /NXCOMPAT:NO does not apply to WinCE.	
Source code analysis	VS2005 Code Analysis Options ("/analyze). For VS2005 code analysis, all warning IDs from the following list must be fixed: 4532 6029 6053 6057 6059 6063 6067 6200 6201 6202 6203 6204 6248 6259 6260 6268 6276 6277 6281 6282 6287 6288 6289 6290 6291 6296 6298 6299 6305 6306 6308 6334 6383		

WINCE REQUIREMENTS: COMPACT FRAMEWORK MANAGED CODE

Compiler/tool	Minimum required version and switches/options	Optimal/recommended version and switches/options	Comments
C# Compiler	Visual Studio 2005		
csc.exe	Version 8.0.50727.42		
.NET Framework	Version 2.0.50727		
FxCop	Most recent version	Most recent version	

APPENDIX F: SDL REQUIREMENT: NO EXECUTABLE PAGES

Executing code from data code pages is a very common attack vector. The use of this technique is needed only in very limited scenarios. As a result, all programs and services should avoid this technique unless explicitly required. Having even one page marked EXECUTABLE in a process (other than dynamic-link libraries or DLLs) usually renders other security measures (such as /GS and /SafeSEH) useless for that process.

GOALS AND JUSTIFICATION

The Windows Exception Handling mechanism assumes that it is safe to dispatch exceptions to any address if they are not in a DLL but are still EXECUTABLE.

Given a stack buffer overflow, an attacker can overflow the nearest exception record on the stack (there is always at least one) and point it to an address in the page marked EXECUTABLE.

Because of the number of stack locations controlled by the overflow at the point the exception handler takes over, many possible op-code sequences would reliably deliver execution back to the attack-supplied buffer. One such sequence is {pop, pop, ret} (possibly interleaved with other instructions). It is also possible to leverage a sequence of op-codes that would produce an arbitrary memory-overwrite in a two-stage attack.

Because of the number of possibilities, it is very hard to prove that bytes on a page marked EXECUTABLE cannot be abused sufficiently to take control.

SCOPE

The following subsections specify the scope of the No Executable Pages requirement proposal.

OPERATING SYSTEMS

This requirement applies to Win32 and Win64 operating systems, but not to Windows CE or Macintosh.

PRODUCTS/SERVICES

This requirement applies to code that runs on customers' computers (products), as well as to code that runs only on Microsoft-owned computers and used by customers (e.g. Microsoft owned and provisioned online services).

TECHNOLOGIES

This requirement applies to both unmanaged (native) code such C and C++ and managed code such as C#.

NEW CODE AND/OR LEGACY CODE

This requirement applies to both new code and legacy code.

EXCEPTIONS

- Sometimes it simply might not be possible to mark all pages as non-executable. Examples include digital rights management technologies and just-in-time (JIT) technologies that dynamically create code. For such cases, the following techniques can help make the pages safe:If your code uses VirtualAllocXXX APIs to allocate EXECUTABLE pages, load a dummy DLL and use one of its sections instead.

- Register a Vectored Exception Handler in the process and vet the chain of exception handlers to make sure none of them point to your EXECUTABLE pages.
- Randomize the address of the EXECUTABLE pages and/or randomize the starting offset of content within those pages.

SPECIAL CASES

Because marking a binary as “DEP compatible” (via /NXCOMPAT) changes how the operating system interacts with the executable, it is important to test all executables (.EXE) marked as /NXCOMPAT with a version of Windows that supports this functionality:

- Client software: Windows XP™ SP2, Windows Vista™
- Server software: Windows Server® 2003 Service Pack 1 (SP1) or Windows "Longhorn" Server

(Review the detailed description of the Data Execution Prevention (DEP) feature for specific details about how to use DEP on Windows Server 2003 SP1.)

All executables (.EXE) marked as /NXCOMPAT will be able to take advantage of Data Execution Protection. Dynamic-link libraries (.DLL files), or other code called by executables (such as COM objects) will not gain direct security benefits via /NXCOMPAT, but will need to coordinate enabling /DEP support with any executable files that might call them. Any EXE with /NXCOMPAT enabled that loads other code without /NXCOMPAT enabled may have the process fail unexpectedly unless the EXE and all of the other code that it calls (DLLs, COM objects, and so on) have been thoroughly tested with DEP enabled (linked with /NXCOMPAT option).

REQUIREMENTS

REQUIREMENT: DO NOT USE CERTAIN VIRTUALALLOCXXX FLAGS

If your code calls any of these APIs:

- VirtualAlloc
- VirtualAllocEx
- VirtualProtect
- VirtualProtectEx
- NtAllocateVirtualMemory
- NtProtectVirtualMemory

Do not use any of these flags:

- PAGE_EXECUTE
- PAGE_EXECUTE_READ
- PAGE_EXECUTE_READWRITE
- PAGE_EXECUTE_WRITECOPY

REQUIREMENT: USE /NXCOMPAT LINKER OPTION

All binaries must link with /NXCOMPAT flag (and not link with /NXCOMPAT:NO) using the linker included with Visual Studio 2005 and later.

COMPLIANCE MEASUREMENT

REQUIREMENT: DO NOT USE CERTAIN VIRTUALALLOCXXX FLAGS

While no solutions exist to monitor the use of these APIs, project teams will be asked to examine project specifications for the use of these APIs and attest to their removal.

REQUIREMENT: USE /NXCOMPAT LINKER OPTION

REQUIREMENT: MEASUREMENT BY PRODUCT/SERVICE TEAM

Project teams will be asked to will be asked to attest to the use of this linker option

REQUIREMENT: MEASUREMENT BY SECURITY ADVISORS

This requirement is required to complete the final security review and use of this linker option should be confirmed before allowing the software to be released to manufacturing or the web (RTM/RTW).

SUPPORT CONSIDERATIONS

REQUIREMENT: IMPACT ON EXISTING SDL REQUIREMENTS

This requirement currently has some overlap with the Banned API requirement in that it describes some function calls that are prohibited. After the release of Windows Vista, this requirement will become an even more important part of the Microsoft defense-in-depth strategy (in conjunction with planned Address Space Layout Randomization (ASLR) support in Windows Vista).

REQUIREMENT: EDUCATION AND TRAINING

The only education reference materials that are currently available are the following:

- Data Execution Prevention
- Detailed description of the Data Execution Prevention (DEP) feature

APPENDIX G: SDL REQUIREMENT: NO SHARED SECTIONS

Binaries that are shipped as part of the product must not contain sections marked as *shared*, which are a security threat and should not be used. Use properly secured, dynamically created shared memory objects instead.

RATIONALE

The Portable Executable (PE) format allows binaries to define sections—named areas of code or data—that have distinct properties, such as size, virtual address, and flags that define the behavior of the operating system as it maps the sections into memory when the binary image is loaded. An example of a section would be *text*, which is typically present in all executable images. This section is used to store the executing code and is marked as Code, Execute, Read, which means code can execute from it, but data cannot be written to it. It is possible to define custom sections with desired names and properties by using compiler/linker directives.

One such section flag is *Shared*. When it is used and the binary is loaded into multiple processes, the shared section will map to the same physical memory address range. This functionality makes it possible for multiple processes to write to and read from addresses that belong to the shared section.

Unfortunately, it is not possible to secure a shared section. Any malicious application that runs in the same session can load the binary with a shared section and eavesdrop or inject data into shared memory (depending on whether the section is read-only or read-write).

To avoid security vulnerabilities, use the *CreateFileMapping* function with proper security attributes to create shared memory objects.

DETECTING EXISTING SHARED PE SECTIONS

You can use the following linker directive to create a shared section:

```
/section:<name>, RWS
```

The following directives in C/C++ source code can be used:

```
// (this introduces a new PE section)
#pragma data_seg(".shared")
            int mySharedData = 1;
#pragma data_seg()
```

or:

```
#pragma section(".shrd2", read, write, shared)
__declspec(allocate(".shrd2")) int mySharedData2 = 2;
```

ADDITIONAL INFORMATION

- Creating Named Shared Memory

APPENDIX H: SDL STANDARD ANNOTATION LANGUAGE (SAL) RECOMMENDATIONS FOR NATIVE WIN32 CODE

The Standard Source Code Annotation Language (SAL), a technology from Microsoft Research that is actively embraced by Windows and Office, is a powerful addition to C/C++ source code to help find bugs, especially security vulnerabilities. SAL can help find more vulnerabilities than the present set of static analysis tools can find. A major benefit of SAL is that developers need only annotate their headers to provide benefit for others. For example, most C runtime and Windows headers that ship with Visual Studio 2005 and later are annotated. Windows Development Kit headers are also annotated.

All products developed using SDL should use a subset of SAL to help find deeper issues such as buffer overrun issues. Microsoft teams such as Office and Windows are using SAL beyond the requirements of SDL.

SAL DETAILS

SAL is primarily used as a method to help tools, such as the Visual C++ /analyze compiler option to find bugs by knowing more about a function interface. For the purposes of this appendix, SAL can document three properties of a function:

- Whether a pointer can be NULL
- How much space can be written to a buffer
- How much can be read from a buffer (potentially including NULL termination)

At a high level, things become more complicated because there are two implementations of SAL:

- __declspec syntax (VS2005 and VS2008)
- Attribute syntax (VS2008)

Each of these implementations maps onto lower-level primitives that are too verbose for typical use. Therefore, developers should use macros that define commonly used combinations in a more concise form. C or C++ should add the following to their precompiled headers:

```
#include “sal.h”
```

SDL RECOMMENDATIONS

- Start with new code only. Microsoft strongly recommends that you also plan to annotate old code.
- All function prototypes that accept buffers in internal header files you create should be SAL annotated.
- If you create public headers, you should annotate all function prototypes that read or write to buffers.

SAL IN PRACTICE

All examples uses the __declspec form.

A classic example is that of a function that takes a buffer and a buffer size as arguments. You know that the two arguments are closely connected, but the compiler and the source code analysis tools do not know that. SAL helps bridge that gap.

A list of common SAL annotations, with examples can be found at:
http://blogs.msdn.com/michael_howard/archive/2006/05/19/602077.aspx

The following code demonstrates this benefit by annotating a writeable buffer named buf:

```
        void FillString(
    TCHAR* buf,
    int cchBuf,
    TCHAR ch) {
    for (int i = 0; i < cchBuf; i++)
      buf[i] = ch;
  }
```

cchBuf is the character count of buf. Adding SAL helps link the two arguments together:

```
  void FillString(
    __out_ecount(cchBuf) TCHAR* buf,
    int cchBuf,
    TCHAR ch) {
    for (int i = 0; i < cchBuf; i++)
      buf[i] = ch;
  }
```

If you compile this code for Unicode, a potential buffer overrun exists when you call this code:

```
  TCHAR buf[MAX_PATH];
  FillString(buf, sizeof(buf), '\0');
```

sizeof is a byte count, not a character count. The programmer should have used countof.

In the __out_ecount macro, __out means the buffer is an "out" buffer and is written to by the functions. The buffer size, in elements, is _ecount(cchBuf). Note that this function cannot handle a NULL buf, if it could, then the following macro could be used: __out_ecount_opt(cchBuf), where _opt means optional.

The following example shows a function that reads to a buffer and writes to another.

```
  void CopyRange(__in_ecount(cchFrom) const char *from,
        size_t cchFrom,
        __out_ecount(cchTo) char *to,
        size_t cchTo);
```

TOOLS USAGE

To take advantage of SAL, make sure you compile your code with version of VC++ 2005 or VC++ 2008 that support the /analyze compile-time flag.

TOP PRIORITY WARNINGS TO TRIAGE FOR FIXING

6029 Possible buffer overrun in call to <function>: use of unchecked value

6053	Call to <function>: may not zero-terminate string <variable>
6057	Buffer overrun due to number of characters/number of bytes mismatch in call to <function>
6059	Incorrect length parameter in call to <function>: pass the number of remaining characters, not the buffer size of <variable>
6200	Index <name> is out of valid index range <min> to <max> for non-stack buffer <variable>
6201	Buffer overrun for <variable>, which is possibly stack allocated: index <name> is out of valid index range <min> to <max>
6202	Buffer overrun for <variable>, which is possibly stack allocated, in call to <function>: length <size> exceeds buffer size <max>
6203	Buffer overrun for buffer <variable> in call to <function>: length <size> exceeds buffer size
6204	Possible buffer overrun in call to <function>: use of unchecked parameter <variable>
6209	Using "sizeof<variable1>" as parameter <number> in call to <function> where <variable2> may be an array of wide characters; did you intend to use character count rather than byte count?
6248	Setting a SECURITY_DESCRIPTOR's DACL to NULL will result in an unprotected object
6383	Buffer overrun due to conversion of an element count into a byte count

BENEFITS OF SAL

Because SAL provides more function interface information to the compiler toolset, SAL will find more bugs earlier and with less noise.

MORE INFORMATION

More detailed SAL information can be found in chapter 1 of Writing Secure Code for Windows Vista from Howard and LeBlanc and at http://blogs.msdn.com/michael_howard/archive/2006/05/19/602077.aspx

SUMMARY

- Microsoft recommends that you start by annotating new code only. As time permits, existing code should be annotated also.
- You should use SAL for all functions that write to buffers.
- You should consider using SAL for all functions that read from buffers.
- The SDL requirement does not mandate either SAL macro syntax. Use attribute or __declspec as you see fit.
- Annotate the function prototypes in headers that you create.
- If you consume public headers, you must use only annotated headers.

APPENDIX I: SDL REQUIREMENT: HEAP MANAGER FAIL FAST SETTING

During the past few years, Microsoft added core defenses at the operating system level to help protect against some types of attacks. None of them are perfect, but when used together they can provide an effective defense. Examples include the firewall, the –GS flag, heap checking, and DEP (also known as NX). Generally, Microsoft introduces an initial version or subset of the defense and then augments it over time as developers and end users become accustomed to it.

A good example is Data Execution Protection (DEP). This feature was never enabled in Microsoft® Windows 2000 because it had no hardware support, but became available in Windows Server® 2003 as an unsupported boot.ini option. DEP was then supported for the first time in Windows XP Service Pack 2 (SP2), but set only for the system and not for non-system applications.

Another example, and the focus of this requirement, is the ability to detect and respond to heap corruption. In the past, there was no protection in the heap from heap-based buffer overruns. Microsoft then added metadata checking, primarily in the form of forward and backward link checking post block-free to determine whether a heap overrun had occurred. However, for application compatibility reasons, the mitigation was limited to preventing the arbitrary write controlled by a potential exploit from taking place, and the application was allowed to continue to run after the point the corruption was detected. Windows Vista™ includes a more robust mechanism: the application terminates when heap corruption is detected. This mechanism also helps developers find and fix heap–based overruns early in the development lifecycle.

This Heap Manager Fail Fast Setting requirement might cause reliability issues in applications that have poor heap memory management. However, the failing code is found immediately and can be fixed, which makes software both more secure and more reliable. Windows Vista has encountered only one such example in a third-party ActiveX® control.

This capability is enabled for some core operating system components, but not for non-system applications running on Windows Vista. This appendix outlines how to enable the option for non-system applications.

GOALS AND JUSTIFICATION

Currently, even if corruption is detected in the heap manager, the process might continue to run successfully, depending on the corruption pattern, the data affected, and the usage. Some applications might hide memory-related bugs by handling exceptions, such as access violations, that are raised inside the heap manager. The goal is to find bugs early and create robust code. Therefore, certain critical operating system components must hard fail when heap corruption is detected. Also, any new application developed and tested on Vista should terminate on heap corruptions.

This requirement has two major benefits:

- The first benefit applies to development: With this requirement in place, problematic code is more likely to be found because the failure is immediate. Think of it as an “assert” on heap overrun. However, the code that performs heap-based memory allocation and manipulation must be tested correctly. The best method to find this class of bug is through fuzz testing.
- The second benefit is in deployment: If a bug is missed during development and a heap-based overrun exploit occurs, the exploit would become a denial-of-service issue rather than a potential code execution bug.

The requirement is a no-op on versions of the Windows operating system prior to Windows Vista because the required setting is ignored.

SCOPE

The following subsections specify the scope of the Heap Manager Fail Test Setting requirement proposal.

OPERATING SYSTEM

This requirement applies only to Win32.

PRODUCTS/SERVICES

This proposal applies to code that runs on customers' computers (products), as well as to code that runs only on Microsoft-owned computers and used by customers (services).

TECHNOLOGIES

This proposal applies to unmanaged (native) code such C and C++ but not to managed code such as C#.

NEW CODE AND/OR LEGACY CODE

This proposal applies to new code but not to legacy code.

EXTERNAL APPLICABILITY

This proposal applies to external third-party ISV code.

EXCEPTIONS AND SPECIAL CASES

There are no exceptions or special cases for new code.

REQUIREMENT DEFINITION

Before you use any heap–based memory, you must add the following code to your application startup:

```
(void)HeapSetInformation(NULL,
HeapEnableTerminationOnCorruption,
NULL,
0);
```

Microsoft also recommends that the code use the Low-Fragmentation Heap, which has been shown to be more resistant to attack than the "normal" heap in Windows. To use the Low-Fragmentation Heap, use the following code:

```
DWORD Frag = 2;
(void)HeapSetInformation(NULL,
HeapCompatibilityInformation,
&Frag,
sizeof(&Frag));
```

You must add these function calls as early as possible on application startup. This requirement applies to all unmanaged .EXE files, but not to dynamic-link libraries (DLLs), which do not need to call this function.

COMPLIANCE MEASUREMENT

A product/service team can verify compliance in either of the two ways mentioned in the following requirement.

REQUIREMENT: MEASUREMENT BY PRODUCT/SERVICE TEAM

- Verify that the correct function is included in the main() function of the product and attest to its use.

or:

- Run the application under a kernel debugger, issue the !heap -s command, and verify that the following text appears: **Termination on corruption : ENABLED**.

APPENDIX J: SDL REQUIREMENT: APPLICATION VERIFIER

Application Verifier is a runtime verification tool for unmanaged code. It helps developers quickly find subtle programming errors that can be extremely difficult to identify with typical application testing. Application Verifier makes it easier to create reliable applications by monitoring an application's interaction with the Microsoft® Windows® operating system. It profiles the application's use of kernel objects, the registry, the file system, and Win32 APIs (heap, handle, locks, and more).

WHY IS APPLICATION VERIFIER IMPORTANT?

Application Verifier can help quickly identify security issues related to heap buffer overruns by enabling it when test scenarios are covered. As a result of using it, your organization could avoid having to release security bulletins related to such problems and save you both money and credibility.

CODE REQUIRED TO RUN APPLICATION VERIFIER

Application Verifier should be run on all unmanaged code.

Application Verifier is a tool that detects errors in a process (user mode software) while the process is running. Typical findings include heap corruptions (including heap buffer overruns) and incorrect synchronizations and operations. Whenever Application Verifier finds an issue, it goes into debugger mode. Therefore, either the application being verified should run under a user-mode debugger or the system should run under a kernel debugger.

APPLICATION VERIFIER USAGE SCENARIOS

Application Verifier (available in Microsoft Visual Studio®) cannot be enabled for a running process. You need to make settings as described in this appendix and then start the application. The settings are persistent until explicitly deleted. Therefore, an application will always start with AppVerifier enabled, regardless of how many times you launch it

The scenarios in this appendix showcase the recommended command-line options for quality gates that you should run during all tests (BVTs, stress, unit, and regression) that exercise the code change.

TESTING WITH APPLICATION VERIFIER

The expectation for this scenario is that the application does not break into debugger mode, and that all tests pass with the same pass rate as when run without Application Verifier enabled.

1. Enable verifier for the application(s) you wish to test using:

   appverif /verify <*MyApp.exe*>

   Note /verify will enable the base checks: HANDLE_CHECKS, RPC_CHECKS, COM_CHECKS, LOCK_CHECKS, FIRST_CHANCE_EXCEPTION_CHECKS, and FULL_PAGE_HEAP.
2. If you are testing a dynamic-link library (DLL), you must enable the verifier for the test .exe that is exercising the DLL.
3. Run ALL your tests exercising the application.
4. Analyze any debugger break that you encounter. Debugger breaks signify bugs found by the verifier, and you will need to understand and fix them.
5. When you are finished, delete all settings made with:

   appverif /n <*MyApp.exe*>

You can debug any issues you find with Application Verifier by reviewing the Verifier Stop codes within the help contents.

TESTING WITH APPLICATION VERIFIER AND FAULT INJECTION

The expectation for this scenario is that the application does not break into debugger mode. Not breaking into the debugger mode means there are no errors that need to be addressed.

The pass rate for the tests may decrease significantly because random fault injections are introduced into the normal operation.

6. Enable verifier and fault injection for the application(s) you wish to test by using the following command-line syntax:

   appverif /verify *<MyApp.exe>* /faults

   Note If you are testing a DLL, you can apply fault injection on a certain DLL instead of on the entire process. The command-line syntax would be:

   appverif /verify TARGET [/faults [PROBABILITY [TIMEOUT [DLL …]]]]

   For example, appverif /verify *<mytest.exe>* /faults 5 1000 d3d9.dll
7. Run *all* your tests exercising the application.
8. Analyze any debugger break that you encounter. Debugger breaks signify bugs found by the verifier, and you will need to understand and fix them.
9. When you are finished, delete all settings made with:

   appverif /n *<MyApp.exe>*

Note that running with and without fault injection exercises different code paths in an application. Therefore, you must run both scenarios to obtain the full benefit of Application Verifier.

APPENDIX K: SDL PRIVACY ESCALATION RESPONSE FRAMEWORK (SAMPLE)

This sample document provides basic criteria to consider when building a privacy breach response process.

PURPOSE

The purpose of the Privacy Escalation Response Framework (PERF) is to define a systematic process that you can use to resolve privacy escalations efficiently. The process must also manage the associated internal and external communications and identify the root cause or causes of each escalation, so that policies and/or processes can be improved to help prevent recurrences.

DEFINITION: PRIVACY ESCALATION

A *privacy escalation* is an internal process to communicate the details of a privacy-related incident. A privacy escalation is warranted for the following types of incidents:

- Data breaches or theft
- Failure to meet communicated privacy commitments
- Privacy-related lawsuits
- Privacy-related regulatory inquiries
- Contact from media outlets or a privacy advocacy group regarding a privacy incident

PRIVACY ESCALATION TEAM

Your privacy escalation core team should include an escalation manager, a legal representative, and a public relations representative at a minimum. The escalation manager should be responsible for including appropriate representation from across your organization (such as privacy and business experts) and for driving the process to completion. The legal and public relations representatives are responsible for helping resolve any legal or PR concerns consistently throughout the process.

SUBMITTING PRIVACY ESCALATION REQUESTS

The privacy core team should set up a distribution group or managed e-mail account that any employee can contact regarding a potential privacy escalation.

PRIVACY ESCALATION RESPONSE PROCESS

Escalation should begin when the first e-mail notification of the issue is received. The escalation manager is responsible for evaluating the content of the escalation to determine whether more information is required. If so, the escalation manager is responsible for working with the reporting party and other contacts to determine:

- The source of the escalation
- The impact and breadth of the escalation
- The validity of the incident or situation
- A summary of the known facts
- Timeline expectations

- Employees who know about the situation, product, or service

The escalation manager should then disseminate this information to appropriate contacts and seek resolution. Although the escalation manager can assign portions of the workload to other people as appropriate, the escalation manager should ensure that all aspects of the escalation are resolved appropriately. Appropriate resolutions should be determined by a privacy escalation core team in cooperation with the reporting party and other applicable contacts. Appropriate resolutions might include some or all of the following:

- Internal incident management
- Communications and training
- Human Resources actions in the case of a deliberate misuse of data
- External communications, such as:
    - Online Help articles
    - Public relations outreach
    - Breach notification
    - Documentation updates
    - Short-term and/or long-term product or service changes

CLOSING

After all appropriate resolutions are in place, the privacy escalation team should evaluate the effectiveness of privacy escalation response actions. An effective remediation is one that resolves the concerns of the reporting party, resolves associated customer concerns, and helps to ensure that similar events will not recur.

APPENDIX L: GLOSSARY

buffer overflow: A condition that occurs because of a failure to check or to limit input data buffer sizes before data is manipulated or processed.

bug bar: A set of criteria that establishes a minimum level of quality.

deprecation: Designating a component for future removal from a software program.

fuzz testing: A means of testing that causes a software program to consume deliberately malformed data to see how the program reacts.

giblets: Code that was created by external development groups in either source or object form.

harden: Take steps to ensure no weaknesses or vulnerabilities in a software program are exposed.

implicit consent: An implied form of consent in certain limited home and organizational networking scenarios.

informed consent: An explicitly stated form of consent that is usually provided after some form of conditions acknowledgment.

penetration testing (pen testing): A test method where the security of a computer program or network is subjected to deliberate simulated attack. See: http://en.wikipedia.org/wiki/Penetration_Testing

personally identifiable information (also known as PII): Data that provides personal or private information that should not be publicly available. Examples include financial or medical information.

port exception: An exception to a firewall policy that specifies a certain logical port in the firewall should be opened or closed.

privacy escalation: An internal process to communicate the details of a privacy-related incident. A privacy escalation is typically warranted for data breaches or theft, failure to meet communicated privacy commitments, privacy-related lawsuits, privacy-related regulatory inquiries, and contact from media outlets or a privacy advocacy group regarding a privacy incident.

privacy impact rating: A measurement of the sensitivity of the data a software program will process from a privacy perspective.

privacy lead or privacy champ: An individual on a software development team who will be responsible for privacy for the software program being developed.

program exception: An exception to a firewall policy that exempts a specific program or programs from some aspect of the policy.

security push: A team-wide focus on threat model updates, code review, testing, and documentation scrub. Typically, a security push occurs after a product is code/feature complete.

service pack: A means by which product updates are distributed. Service packs might contain updates for system reliability, program compatibility, security, or privacy. A service pack requires a previous version of a product before it can be installed and used. A service pack might not always be named as such; some products may refer to a service pack as a *service release*, *update*, or *refresh*.

zero-day exploit: An exploit of a vulnerability for which a security update does not exist.

APPENDIX M: SDL PRIVACY BUG BAR (SAMPLE)

Note: This sample document is for illustration purposes only. The content presented below outlines basic criteria to consider when creating privacy processes. It is not an exhaustive list of activities or criteria, and should not be treated as such.

Please refer to the definitions of terms section below.

End User Scenarios Usage Notes: • These scenarios apply to Consumers, Enterprise Clients, and Enterprise Administrators acting as end users. For Enterprise Administrators acting in their administrative role, see the Enterprise Administrators Scenarios.	
SEV 1	• Lack of Notice & Consent: ○ Example – ▪ Transfer of sensitive Personally Identifiable Information PII from the user's system without Prominent Notice & Explicit Opt-in Consent in the UI prior to transfer. • Lack of User Controls: ○ Example – ▪ Ongoing collection and transfer of non-essential PII without the ability within the UI for the user to stop subsequent collection and transfer. • Lack of Data Protection: ○ Example – ▪ PII is collected and stored in a persistent general database without an authentication mechanism for users to access and correct stored PII. • Lack of Child Protection: ○ Example – ▪ Age is not collected for a site or service that is attractive to or directed at children and the site collects, uses, or discloses the user's PII. • Improper Use of Cookies: ○ Example – ▪ Sensitive PII stored in a cookie is not encrypted. • Lack of Internal Data Management and Control: ○ Example – ▪ Access to PII stored at Organization is not restricted only to those who have a valid business

	need or there is no policy to revoke access after it is no longer required. • Insufficient Legal Controls: ○ Example – ▪ Product or feature transmits data to an agent or independent third party that has not signed a legally approved contract.
SEV 2	• Lack of Notice & Consent: ○ Example – ▪ Transfer of non-Sensitive PII from the user's computer without Prominent Notice & Explicit Opt-in Consent in the UI prior to transfer. • Lack of User Controls: ○ Example – ▪ Ongoing collection and transfer of non-essential Anonymous Data without the ability in the UI for the user to stop subsequent collection and transfer. • Lack of Data Protection: ○ Example – ▪ Persistently stored non-Sensitive PII lacks a mechanism to prevent unauthorized access. A mechanism is not required where the user is notified in the UI that data will be shared (e.g., Folder labeled "Shared"). • Data Minimization: ○ Example – ▪ Sensitive PII transmitted to an independent third party is not necessary to achieve the disclosed business purpose. • Improper Use of Cookies: ○ Example – ▪ Non-Sensitive PII stored in a persistent cookie is not encrypted.
SEV 3	• Lack of User Controls: ○ Example – ▪ PII is collected and stored locally as Hidden Metadata without any means for a user to remove the metadata. PII is accessible by others, or may be transmitted if files or folders are shared. • Lack of Data Protection: ○ Example –

	▪ Temporarily stored non-Sensitive PII lacks a mechanism to prevent unauthorized access during transfer or storage. A mechanism is not required where the sharing of information is obvious (e.g., user name) or there is Prominent Notice. • Data Minimization: ○ Example – ▪ Non-Sensitive PII or Anonymous Data transmitted to an independent third party is not necessary to achieve disclosed business purpose. • Improper Use of Cookies: ○ Example – ▪ Use of persistent cookie where a session cookie would satisfy the purpose. Or, persisting a cookie for a period that is longer than necessary to satisfy the purpose. • Lack of Internal Data Management and Control: ○ Example – ▪ Data stored at Organization does not have a retention policy.
SEV 4	• Lack of Notice & Consent: ○ Example – ▪ PII is collected and stored locally as Hidden Metadata without Discoverable Notice. PII is not accessible by others, and is not transmitted if files or folders are shared.

Enterprise Adminstration Scenarios Usage Notes: • These scenarios apply to Enterprise Administrators acting in their administrative role. For Enterprise Administrators in an end user role, see the End User Scenarios.	
SEV 1	• Lack of Enterprise Controls: ○ Example – ▪ Automated data transfer of Sensitive PII from the user's system without Prominent Notice and Explicit Opt-in Consent in the UI from the Enterprise Administrator prior to transfer. • Insufficient Privacy Disclosure: ○ Example – ▪ Deployment or Development Guide for Enterprise Administrators provides legal advice.

SEV 2	• Lack of Enterprise Controls: ○ Example – ▪ Automated data transfer of non-Sensitive PII, or Anonymous Data from the user's system without Prominent Notice and Explicit Opt-in Consent in the UI from the Enterprise Administrators prior to transfer. Notice and consent must appear in the UI not via the End-User License Agreement (EULA) or Terms of Service. • Insufficient Privacy Disclosure: ○ Example – ▪ Disclosure to Enterprise Administrators such as Deployment Guide or UX does not disclose storage or transfer of PII.
SEV 3	• Lack of Enterprise Controls: ○ Example – ▪ No mechanism is provided or identified to help the Enterprise Administrators prevent accidental disclosure of User Data (e.g., set site permissions).

Definition of terms

Anonymous Data – Non-Personal data which has no connection to an individual. By itself, it has no intrinsic link to an individual user. For example, hair color or height (in the absence of other correlating information) does not identify a user.

Child or Children – Under 14 in Korea and under 13 in the US.

Discoverable Notice – A "Discoverable Notice" is one the user has to find (e.g., by locating and reading a Privacy Statement of a Web site or by selecting a Privacy Statement link from a Help menu).

Discrete transfer – Data transfer is discrete when it is an isolated data capture event that is not ongoing.

Essential Metadata – Metadata that is necessary to the application for supporting the file (e.g., File extension)

Explicit Consent – "Explicit Consent" requires that the user take or have the ability to take an explicit action before data is collected or transferred.

Hidden Metadata – "Hidden Metadata" is information that is stored with a file but is not visible to the user in all views. Hidden data may include personal information or information that the user would likely not want to distribute publicly. If such information is included, the user must be made aware that this information exists and must be given appropriate control over sharing it.

Implicit Consent – "Implicit Consent" does not require an explicit action indicating consent from the user; the consent is implicit in the operation the user initiates.

Non-Essential Metadata – Metadata that is not necessary to the application for supporting the file (e.g., Key Words)

Persistent storage – Persistent storage of data means that the data continues to be available after the user exits the application.

PII – Personally Identifiable Information is any information (i) that identifies or can be used to identify, contact, or locate the person to whom such information pertains, or (ii) from which identification or contact information of an individual person can be derived. Personally Identifiable Information includes, but is not limited to: name, address, phone number, fax number, email address, financial profiles, medical profile, social security number, and credit card information. Additionally, to the extent unique information (which by itself is not PII such as a unique identifier or IP address) is associated with PII, then such unique information will also be considered Personally Identifiable Information.

Prominent Notice – A "Prominent Notice" is one that is designed to catch the user's attention. Prominent Notices should contain a high-level, substantive summary of the privacy-impacting aspects of the feature such as what data is being collected and how that data will be used. The summary should be fully visible to a user without additional action on the part of the user, such as having to scroll down the page. Prominent Notices should also include clear instructions for where the user can get additional information (such as in a privacy statement).

Sensitive PII – Sensitive Personally Identifiable Information includes any data that could (i) be used to discriminate (e.g., ethnic heritage, religious preference, physical or mental health), (ii) facilitate identity theft (like mother's maiden name), or (iii) permit access to a user's account (like passwords or PINs). Note that if the data described in this paragraph is not commingled with PII during storage or transfer, and it is not correlated with PII, then the data can be treated as Anonymous Data. If there is any doubt, however, the data should be treated as Sensitive PII. While not technically Sensitive PII, User Data that makes users nervous (such as real-time location) should be handled in accordance with the rules for Sensitive PII.

> **SEV 1** – *Release may create legal or regulatory liability for Organization.*
>
> **SEV 2** – *Release may create high risk of negative reaction by privacy advocates, damage Organization's image.*
>
> **SEV 3** – *Some customer concerns may be raised, some privacy advocates may question, but repercussion will be limited.*
>
> **SEV 4** – *May cause some customer queries. Scrutiny by privacy advocates unlikely.*

Temporary storage – Temporary storage of data means that the data is only available while the application is running.

APPENDIX N: SDL SECURITY BUG BAR (SAMPLE)

Note: This sample document is for illustration purposes only. The content presented below outlines basic criteria to consider when creating security processes. It is not an exhaustive list of activities or criteria, and should not be treated as such.

Please refer to the Please refer to the definitions of terms section below.

Server Please refer to the Denial of Service Matrix below for a complete matrix of server DoS scenarios.	
SEV 1	• Elevation of Privilege – The ability to either execute arbitrary code OR obtain more privilege than intended ○ Remote Anonymous User
SEV 2	• Denial of Service ○ Must be “easy to exploit” by sending small amount of data or be otherwise quickly induced ○ Anonymous • Elevation of Privilege – The ability to either execute arbitrary code OR obtain more privilege than intended ○ Remote Authenticated User ○ Local Authenticated User (Terminal Server) • Information Disclosure (Targeted) ○ Cases where the attacker can locate and read information *from anywhere* on the system, including system information, that was not intended/designed to be exposed • Spoofing ○ Computer connecting to server is able to masquerade as a different user or computer of his/her choice *using a protocol* that is designed and marketed to provide strong authentication • Tampering ○ Permanent modification of any user data or data used to make trust decisions *in a common or default scenario* that persists after restarting the OS/application.
SEV 3	• Denial of Service ○ Anonymous ▪ Temporary DoS without amplification in a default/common install ○ Authenticated ▪ Permanent DoS ▪ Temporary DoS with amplification in a default/common install • Information Disclosure (Targeted) ○ Cases where the attacker can easily read information on the system *from known locations*,

	including system information, that was not intended/designed to be exposed • Spoofing ○ Client user or computer is able to masquerade as a different, random user or computer using a protocol that is designed and marketed to provide strong authentication • Tampering ○ Permanent modification of any user data or data used to make trust decisions *in a specific scenario* that persists after restarting the OS/application. ○ Temporary modification of data *in a common or default scenario* that does not persist after restarting the OS/application.
SEV 4	• Information Disclosure (Untargeted) ○ Runtime information • Tampering ○ Temporary modification of data *in a specific scenario* that does not persist after restarting the OS/application.
Defense In Depth	• Bug with security ramifications but without any known method for exploit • Design Change Request that reduces attack surface area • Circumvention of a broad security mitigation meant to lower the risk of specific classes of attacks • Information Disclosure: Avoid "Search engine fingerprinting" – presenting data in a way that search engines easily catalog installations of particular products ○ Example: internet facing product has default Web pages but not a "robots.txt" file or other metadata to keep the pages from being cataloged by major search engines • Vulnerabilities that could only be exploited if other external dependencies fail. ○ Examples: ▪ A vulnerability that can't be exploited unless another product has a buffer overrun. • Critical and/or Important vulnerabilities should ideally have multiple mitigations such that if a single mitigation fails, attackers still can't exploit the critical/important vulnerability.

Client Extensive user action is defined as: • "User interaction" can only happen in client-driven scenario. • Normal, simple user actions like previewing mail, viewing local folders or file shares are not "extensive user interaction." • Extensive: User manually navigating to particular web site (ex: typing in URL) or clicking through yes/no decision. • NOT Extensive: User clicking through email links.	
SEV 1	• Elevation of Privilege (Remote) – The ability to either execute arbitrary code OR obtain more privilege than intended

SEV 2	• Elevation of Privilege (Remote) 　○ Execution of Arbitrary code – *with* extensive user action • Elevation of Privilege (Local) 　○ Local low privilege user can elevate themselves to another user, administrator, and/or local system • Information Disclosure (Targeted) 　○ Cases where the attacker can locate and read information on the system, including system information, that was not intended/designed to be exposed • Denial of Service 　○ System Corruption DOS – requires re-installation of system and/or components • Spoofing 　○ Ability for attacker to present UI that is different from but visually identical to UI which users *must rely on to make valid trust decisions* in a *default/common scenario*. A trust decision is defined as any time the user takes an action believing some information is being presented by a particular entity, either the system or some specific local or remote source. • Tampering 　○ Permanent modification of any user data or data used to make trust decisions in a common or default scenario that persists after restarting the OS/application.
SEV 3	• Denial of Service 　○ Permanent DOS – Requires cold reboot or causes Blue Screen/Bug Check • Information Disclosure (Targeted) 　○ Cases where the attacker can read information on the system *from known locations*, including system information, that was not intended/designed to be exposed • Spoofing 　○ Ability for attacker to present UI that is different from but visually identical to UI that users *are accustomed to trust* in *a specific scenario*. "Accustomed to trust" is defined as anything a user is commonly familiar with based on normal interaction with the OS/application but does not typically think of as a "trust decision".
SEV 4	• Denial of Service 　○ Temporary DOS – Requires restart of application. • Spoofing 　○ Ability for attacker to present UI that is different from but visually identical to UI *that is a single part of a bigger attack scenario*. • Tampering 　○ Temporary modification of any data that does not persist after restarting the OS/application. • Information Disclosure (Untargeted)
Defense In	• Clear Bug with security ramifications but without exploit

Depth	• Design Change Requests that reduces attack surface area • Circumvention of a broad security mitigation meant to lower the risk of specific classes of attacks

Definition of terms

Authenticated – Any attack which has to include authenticating by the network. This implies that logging of some type must be able to occur so that the attacker can be identified.

Anonymous – Any attack which does not need to authenticate to complete.

Client – Either software that runs locally on a single computer or software that accesses shared resources provided by a server over a network.

Default/Common – Any features that are active out of the box or that reach more than 10% of users.

Scenario – Any features which require special customization or use cases to enable, reaching less than 10% of users.

Server – Computers that are configured to run software that await and fulfill requests from client processes that run on other computers.

SEV 1 – *A security vulnerability that would be rated as having the highest potential for damage.*

SEV 2 – *A security vulnerability that would be rated as having significant potential for damage, but less than SEV 1.*

SEV 3 – *A security vulnerability that would be rated as having moderate potential for damage, but less than SEV 2.*

SEV 4 – *A security vulnerability that would be rated as low potential for damage.*

Targeted Information Disclosure – Ability to intentionally select (target) desired information.

Temporary DoS – A temporary DoS is a situation where the following criteria are met:

- The target cannot perform normal operations due to an attack.
- The response to an attack is roughly the same magnitude as the size of the attack.
- The target returns to the normal level of functionality shortly after the attack is finished. The exact definition of "shortly" should be evaluated for each product.

For example, a server is unresponsive while an attacker is constantly sending a stream of packets across a network, and the server returns to normal a few seconds after the packet stream stops.

Temporary DoS with Amplification – A temporary DoS with amplification is a situation where the following criteria are met:

- The target cannot perform normal operations due to an attack.
- The response to an attack is magnitudes beyond the size of the attack.
- The target returns to the normal level of functionality after the attack is finished, but it will take some time (perhaps a few minutes).

For example, if I can send a malicious 10 byte packet and cause a 2048k response on the network, then we are DoS'ing the bandwidth by amplifying our attack effort.

Permanent DoS – A permanent DoS is one which requires an admin to start, restart, or reinstall all or parts of the system. Any bug that automatically restarts the system is also a permanent DoS.

Denial of Service (Server) matrix

Authenticated vs. Anonymous attack	Default/Common vs. Scenario	Temporary DoS vs. Permanent	Rating
Authenticated	Default/Common	Permanent	Moderate
Authenticated	Default/Common	Temporary DoS w/ amplification	Moderate
Authenticated	Default/Common	Temporary DoS	Low
Authenticated	Scenario	Permanent	Moderate
Authenticated	Scenario	Temporary DoS w/ amplification	Low
Authenticated	Scenario	Temporary DoS	Low
Anonymous	Default/Common	Permanent	Important
Anonymous	Default/Common	Temporary DoS w/ amplification	Important
Anonymous	Default/Common	Temporary DoS	Moderate
Anonymous	Scenario	Permanent	Important
Anonymous	Scenario	Temporary DoS w/ amplification	Important
Anonymous	Scenario	Temporary DoS	Low

APPENDIX O: SECURITY PLAN (SAMPLE)

Note: This sample document is for illustration purposes only. The content presented below outlines basic criteria to consider when creating security processes. It is not a complete list of activities or criteria, and should not be treated as such.

This document outlines the security activities for <SAMPLE> as they relate to each step of the Security Development Lifecycle (SDL). It describes the objective and provides a basic outline of each activity. It also identifies owners and expected deliverables from the activity. Most of the deliverables are included as exit criteria for different milestones for the project.

For successful execution of this security plan the security team must perform security sign-offs, reviews, check-pointing etc. for the security ship-readiness of the product at various project milestones. It is recommended that a “virtual” team be created – comprised of individuals from program management, development, test, and user education.

The remainder of this document describes the minimum activities needed to build a secure product, the milestones during which they should be performed and the owners and the deliverables for each activity.

The following table describes the SDL processes and code deliverables:

SDL Stage	SDL Book Page	Steps and deliverables
Stage 0: Education and Awareness	55	All members have taken Basics of Secure Design, Development & Testing class, or other approved security classes
Stage 1: Project Inception	67	Determine SDL Applicability Assign Security Advisor Build the Security Leadership Team Determine the Security “Bug Bar” and bug database
Stage 2 : Define and Follow Design Best Practices	75	Define Attack Surface Define Secure Design Principles applicability
Stage 3: Product Risk Assessment	93	Perform Security and Privacy Risk Assessment
Stage 4: Risk Analysis	101	Build Threat Models
Stage 5: Creating Security Docs and Tools	133	Define security best practice documentation and tools
Stage 6: Secure Coding Policies	143	Define and follow compiler tools best practice Choose analysis tools Determine compiler flags and banned functionality

Stage 7: Secure Testing Policies	153	Define fuzz testing, penetration, and run-time tests Re-evaluate attack surface Re-evaluate threat models
Stage 8: The Security Push	169	Determine if a security push is needed Define steps in the security push
Stage 9: Final Security Review	181	Re-evaluate threat models and unfixed security bugs Review tools use Handle exceptions
Stage 10: Security Response Planning	187	Security response process & owners identified. Security Emergency Response Plan completed
Stage 11: Product Release	215	N/A
Stage 12: Security Response Execution	217	N/A

STAGE 0: EDUCATION AND AWARENESS

EDUCATION

- Define which types of security training are available for your personnel.
- Identify who creates and/or delivers the classes.
- Define where they can find information about the classes.

STAGE 1: PROJECT INCEPTION

DETERMINE SDL APPLICABILITY

- Determine whether the SDL applies to your component.

ASSIGN SECURITY ADVISOR

- When your company has a central security leadership team, assign a person from the leadership team to be that team's security advisor.
- When your company has no central security leadership team, assign a "security team lead" from the development team.

BUILD THE SECURITY LEADERSHIP TEAM

- Identify the members of the leadership team.
- Define the methods for communicating with the leadership team.

DETERMINE THE SECURITY "BUG BAR" AND BUG DATABASE

- Define where and how to log and track security bugs.
- Define how security bugs will be triaged.
- Define the location of the security "bug bar" document.

STAGE 2: DEFINE AND FOLLOW DESIGN BEST PRACTICES

DEFINE ATTACK SURFACE

- Be familiar with the attack surface elements that you will enumerate (such as open ports, services running by default etc.)
- Determine the attack surface baseline.
- Define the plan to reduce attack surface.

DEFINE SECURE DESIGN PRINCIPLES APPLICABILITY

- Understand which core security design principles you will explicitly address in this component and why.

STAGE 3: PRODUCT RISK ASSESSMENT

PERFORM SECURITY AND PRIVACY RISK ASSESSMENT

- Identify the author of the “Security and Privacy Risk Assessment” document.
- Define where the “Security and Privacy Risk Assessment” document resides.

STAGE 4: RISK ANALYSIS

BUILD THREAT MODELS

- Define which tool or template you are using to build your threat models.
- Identify where the threat models reside.
- Identify reviewers for the threat models.

STAGE 5: CREATING SECURITY DOCS AND TOOLS

DEFINE SECURITY BEST PRACTICE DOCUMENTATION AND TOOLS

- Define which administrative security best practices documents will be created.
- Define which administrative security best practices tools will be created.

STAGE 6: SECURE CODING POLICIES

DEFINE AND FOLLOW COMPILER TOOLS BEST PRACTICE

- Determine which compilers and compiler versions you are using.
- Discover if your compilers are the latest versions available from the vendor. If they are not, find out why not.

CHOOSE ANALYSIS TOOLS

- Determine which static analysis tools you will use.
- Determine how often they will be run on the source code.

DETERMINE COMPILER FLAGS

- Define which compiler and linker flags will be used.
- Find out if there are deviations from the SDL requirements, and if so, why.

DETERMINE BANNED FUNCTIONALITY

- Determine whether there are any function calls or cryptographic functionality to be banned.
- Find out if there are deviations from the SDL requirements, and if so, why.

- Define how you will enforce the banned functionality policy.

SECURE CODING CHECKLIST

- Identify where developers can find the security coding checklist.
- Define the policy for updating the checklist.
- Determine how often the checklist is updated.

STAGE 7: SECURE TESTING POLICIES

DEFINE FUZZ TESTING, PENETRATION, AND RUN-TIME TESTS

- Determine which file formats will be fuzz tested.
- Determine which networking interfaces will be fuzz tested.
- Determine which tools will be used to fuzz test files and networking interfaces.

RE-EVALUATE ATTACK SURFACE

- Determined when the product attack surface will be re-evaluated.
- Define which variations from the baseline you will tolerate.

RE-EVALUATE THREAT MODELS

- Determine when all threat models will be re-evaluated.
- Define which variations from the baseline you will tolerate.

STAGE 8: THE SECURITY PUSH

DETERMINE IF A SECURITY PUSH IS NEEDED

- Determine if there is a need for a security push.

DEFINE THE SECURITY PUSH STEPS

- Determine the timeline for the security push.
- Determine whether there will be intensive education of project staff prior to the push.
- Determine whether there are any other intensive tasks you want to focus on.
- Define how the security bugs will be tracked.
- Determine whether you will take any further steps beyond SDL.

STAGE 9: FINAL SECURITY REVIEW

RE-EVALUATE THREAT MODELS

- Re-evaluate all threat models and determine their quality as defined in the SDL book.

REVIEW UNFIXED SECURITY BUGS

- Re-evaluate any unfixed security bugs to make sure none have been triaged incorrectly.

REVIEW TOOLS USE

- Make sure all appropriate tools and compiler technologies are using the correct versions and options.

HANDLE EXCEPTIONS

- Find out what the process is for handling any exceptions found (SDL violations), if there are any.

STAGE 10: SECURITY RESPONSE PLANNING

SECURITY RESPONSE PROCESS AND OWNERS IDENTIFIED.

- Define and document the security response process.
- Find out the identity of the security response owners.
- Define how internal personnel communicate with the security response team.
- Define how external people communicate with the security response team.

STAGE 11: PRODUCT RELEASE

- Define the official product sign-off criteria.
- Find out if support personnel require debug symbols, and if so, know their location.

STAGE 12: SECURITY RESPONSE EXECUTION

- No tasks required.