MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

COMPILER DEFENSES (LEVEL 300)

Guide

Version 1.0

The following documentation provides presenter’s notes for the Microsoft Security Development Lifecycle (SDL) Compiler Defenses (Level 300) presentation.

For the latest information, please see [http://www.microsoft.com/sdl](http://go.microsoft.com/?linkid=9672761).

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form, by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, SQL Server, Visio, Visual Basic, Visual C#, Visual C++, Visual Studio, Windows, Windows Server, and Windows Vista are trademarks of the Microsoft group of companies.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

[1.0 Security Development Lifecycle Content 2](#_Toc233605201)

[1.1 Introduction 2](#_Toc233605202)

[1.2 System Requirements 2](#_Toc233605203)

[1.3 Presentation Themes 2](#_Toc233605204)

[2.0 Microsoft SDL Compiler Defenses 2](#_Toc233605205)

[Overview 2](#_Toc233605206)

[Presentation Transcript 3](#_Toc233605207)

[Presentation Voiceover 3](#_Toc233605208)

[Presentation Demonstrations 3](#_Toc233605209)

[Slide 2 – Title Slide 3](#_Toc233605210)

[Slide 3 – Agenda 3](#_Toc233605211)

[Slide 4 – Microsoft Security Development Lifecycle (SDL) 3](#_Toc233605212)

[Slide 5 – Compiler Defenses Overview 4](#_Toc233605213)

[Slide 6 – Microsoft Compiler Defenses Illustrated 5](#_Toc233605214)

[Slide 7 – Buffer Security Checks (/GS) 5](#_Toc233605215)

[Slide 8 – Defending Return Addresses with Security Cookies 6](#_Toc233605216)

[Slide 9 – Defending Return Addresses with Security Cookies 6](#_Toc233605217)

[Slide 10 – Buffer Security Check Demonstration 7](#_Toc233605218)

[Slide 11 – Buffer Security Check Limitations 7](#_Toc233605219)

[Slide 12 – Safe Exception Handling (/SAFESEH) 7](#_Toc233605220)

[Slide 13 – Hardware-Enforced Data Execution Prevention (/NXCOMPAT) 8](#_Toc233605221)

[Slide 14 – Image Randomization 8](#_Toc233605222)

[Slide 15 – Image Randomization Illustration 8](#_Toc233605223)

[Slide 16 – MIDL Compiler (/robust) 9](#_Toc233605224)

[Slide 17 – Microsoft SDL Compiler Defenses Windows 32-bit Requirements 9](#_Toc233605225)

[Slide 18 – Conclusion 9](#_Toc233605226)

[Slide 19 - Appendix 10](#_Toc233605227)

[Slide 20 – Microsoft Security Development Lifecycle (SDL) 10](#_Toc233605228)

[Slide 21 – Microsoft Writing Secure Code Book Series 10](#_Toc233605229)

[Slide 22 – Microsoft Developer Network (MSDN) Security Developer Center 10](#_Toc233605230)

[Slide 23 – Secure Development Blogs 10](#_Toc233605231)

[Slide 24 – Hunting Security Bugs 10](#_Toc233605232)

[Slide 25 – Additional SDL Training 10](#_Toc233605233)

[Slide 26 – Microsoft Compiler Defenses Resources 11](#_Toc233605234)

# 1.0 Security Development Lifecycle Content

## 1.1 Introduction

“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, Windows Server (2003 and 2008) 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 Microsoft SDL 3.2 Whitepaper](http://go.microsoft.com/?linkid=9672762)

To help promote the adoption and awareness of the Microsoft SDL, Microsoft is developing content and demonstrations specifically for external developer audiences. The remainder of this document provides individuals who will present this content internally within their respective organizations with a transcript for the Microsoft SDL Training:

* Compiler Defenses (Level 200) presentation.

## 1.2 System Requirements

In order to use this content, a system that is capable of running [Microsoft PowerPoint 2003](http://www.microsoft.com/powerpoint) or later is required.\

## 1.3 Presentation Themes

The Microsoft PowerPoint deck that accompanies this Presenter’s Guide has been intentionally provided with very limited graphics and formatting. The Microsoft PowerPoint presentation materials have been designed in this fashion to enable individuals who will present this content internally within their respective organizations to incorporate the content into custom PowerPoint themes, styles, and templates with minimal required effort.

# 2.0 Microsoft SDL Compiler Defenses

## Overview

The newest versions of the Microsoft compilers can be used to add defense measures to compiled code. All the necessary defensive code is applied to the application by the compiler, and not by the developer. This means that leveraging Microsoft compiler protection requires minimal effort from application development teams.

This presentation provides an overview of the Microsoft compiler defenses and the related requirements and recommendations from the Microsoft SDL.

The insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on compiler defenses, are provided as a way for external developer communities to enhance its application development practices and the security of its applications.

## Presentation Transcript

This Presentation Transcript section provides a transcript for each slide contained in the Compiler Defenses (Level 300) presentation. The precise transcript text provided herein is also incorporated into the notes section of each slide in the Microsoft PowerPoint Compiler Defenses (Level 300) presentation for ease of reference.

## Presentation Voiceover

A voiceover of the Compiler Defenses (Level 300) presentation transcript below, approximately 30 minutes in length, is also available to assist the presenter in becoming sufficiently acclimated with the subject matter addressed in the Compiler Defenses (Level 300) presentation, as well as to better understand the author’s perspective behind each slide in the presentation.

## Presentation Demonstrations

This presentation uses the Microsoft Virtual Labs environment to facilitate demonstrations in this presentation. Please refer to the following link for further instructions:

[MSDN Virtual Lab: Microsoft SDL Developer Starter Kit: Compiler Defenses](http://go.microsoft.com/?linkid=9672755)

### Slide 2 – Title Slide

The Microsoft Compiler Defenses (Level 300) presentation introduces the role that the Microsoft Security Development Lifecycle (SDL) fulfills in trusted application development. It also provides an overview of the run-time defenses provided by current Microsoft compilers.

Addressing this subject matter will enable our organization to enhance our application development practices and the security of our applications.

Note: It is recommended that the *Buffer Overflows (Level 300)* presentation be reviewed / addressed prior to this presentation.

### Slide 3 – Agenda

In this presentation, we will complete an overview of the Microsoft SDL, key Microsoft compiler defenses and the Microsoft SDL compiler defenses requirements and recommendations.

### Slide 4 – Microsoft Security Development Lifecycle (SDL)

The Microsoft SDL is a holistic and comprehensive approach that leverages education, process, technology and executive commitment to consistently create more secure software internally within and external of Microsoft. Since 2004, all internal Microsoft developers have been required to adhere to the SDL, and Microsoft has updated the SDL every six (6) months to address any emerging threats since its inception.

True to its name, the SDL was created to complement (rather than disrupt) the software development life cycle. The core phases and principles of the SDL include:

**Training phase:** Every Microsoft developer must complete mandatory security training focusing on secure application development practices. Training session topics include topics, such as threat modeling, secure development and testing practices, and security for application development managers.

**Requirements phase:** Requirements for security and privacy must accompany functional requirements of the software that is being created. Such requirements may include the use of encryption, authentication, and other security measures based on the business requirements, exposure and sensitive data. To that end, a security and privacy risk analysis is performed at this stage. In addition, the threshold for security and privacy (or “bug-bar”) is defined during this phase to ensure that vulnerabilities with certain severity are addressed and resolve before the software is officially released.

**Design phase:** Eradicating coding issues with security implications is not sufficient. Design vulnerabilities can have a substantial detrimental impact on security and are much more difficult to address during the verification phase. To that end, threat modeling is a critical SDL requirement and a Microsoft security innovation that is recognized by analysts as the next evolution in creating more secure software. Through threat modeling, architects and developers at Microsoft are able to approach security in a structured and methodical way from an attacker’s perspective. This allows Microsoft to identify and reduce the attack surface and mitigate the risk of potential security design issues.

**Implementation phase:** This is the application code development phase where code is written by developers using industry best practices and analyzed with both internal and externals tools (such as static code analyzers and special security debuggers) to help ensure that those best practices are being followed. Requirements are also specified by the SDL in this phase to ensure that applications are built using the latest compilers versions and built-in compiler protection features.

**Verification phase:** This is the quality assurance phase within which rigorous security testing is conducted in addition to typical functional testing procedures.

**Release phase:** The final security review is the major milestone that a Microsoft product team must pass in order to release a product under the SDL. During this meeting, security experts and the development team review all of the activities, mitigations and security artifacts that are relevant to the project in order to ensure that the security quality requirements are satisfied. During this phase, the product team defines a response plan describing procedures, accountabilities and contact information in case security vulnerabilities are discovered after the product is operational and used by customers.

**Response phase:** After an application is released, the Microsoft Security Response Center (MSRC) handles any security issues that are uncovered “in the wild” and mobilize product teams within Microsoft to provide timely fixes for security issues.

In summary, secure software development requires executive commitment, ongoing process improvement, education and training (from VPs to product managers to developers to testers), tools to aid in detecting security vulnerabilities, and incentives and consequences to ensure everyone adheres to the Microsoft SDL process.

As was previously indicated, this presentation focuses on key Microsoft compiler defenses and how they can be used to make applications more resilient to attack. With respect to specific phases of the Microsoft SDL, this presentation focuses on the Implementation phase.

### Slide 5 – Compiler Defenses Overview

Current Microsoft compilers can be used to add defensive measures to compiled code. Most of the Microsoft compiler defenses are relevant to applications developed in C and C++. Furthermore, Microsoft compiler defense mechanisms are primarily aimed towards limiting risk from buffer overflows and other vulnerabilities that can facilitate a malicious user to control the execution flow of an application.

All the necessary defensive code is applied to the application by the compiler and not by the developer. This means that leveraging Microsoft compiler defenses requires minimal effort from application development teams.

It is important to note that the Microsoft compiler defenses do not fix security vulnerabilities. Applying compiler defenses does not provide a ‘silver bullet’ that will fix or correct security vulnerabilities that may already be present in an application; however, they do provide additional barriers that make exploiting vulnerabilities by malicious users much more difficult. Furthermore, it is entirely possible for each of the defenses discussed in this presentation to be defeated. Again, they each make successfully exploiting application vulnerabilities much more difficult. Compiler defenses should never be used in place of secure application development best-practices.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on compiler defenses, are being shared with each of you as a way for our organization to enhance our application development practices and the security of our applications.

### Slide 6 – Microsoft Compiler Defenses Illustrated

This slide provides an illustration of the run-time defenses applied to applications using Microsoft compilers. Let’s take a look at the application compilation process and then see where compiler defenses can help make applications more resilient to certain attacks.

(Mouse click)

Whenever an application is implemented, it is first implemented as source code using a programming language. Some such programming languages might include, but are not limited to C, C++, and C#.

(Mouse click)

That source code is then inputted into a compiler, which compiles the code into binary objects. A linker then takes and transforms the binary objects into a binary file, such as a library or executable.

(Mouse click)

When compiler defenses are enabled, the compiler and linker apply defensive mechanisms that make the application more resilient to certain attacks.

(Mouse click)

When the application is deployed and is subjected to attack, the run-time defenses will help safely deflect certain attacks.

(Mouse click)

However, it is important to note that compiler defenses will not prevent all types of attacks. Certain attacks may still be able to overcome current compiler defenses. This is why the Microsoft SDL recommends using a layered defense-in-depth approach by implementing a variety of different defenses when developing safer and more trustworthy applications.

Let’s now take a closer look at some of the run-time defenses provided by Microsoft compilers.

### Slide 7 – Buffer Security Checks (/GS)

The first Microsoft compiler defense that will be discussed is the Buffer Security Checks (/GS) option. This compiler defense has been available since Microsoft Visual Studio .NET 2002 and injects security checks into the application code to detect and prevent certain stack-based buffer overflow attacks that overwrite the saved return addresses.

Detection and prevention is accomplished by the use of security cookies to protect critical memory data through rearranging parts of the stack frame and by adding code to protect vulnerable arguments passed to a function. All of these defenses make executing stack-based buffer overflow attacks against /GS-protected applications much more difficult.

To enable buffer security checks, applications must be compiled using the /GS flag, which is enabled by default by the Microsoft compiler.

### Slide 8 – Defending Return Addresses with Security Cookies

The /GS flag is able to provide a key defensive feature through the use of security cookies. Security cookies are sometimes referred to as “canaries;” however, this terminology is more frequently used when discussing similar protection schemes provided by non-Microsoft platforms. Security cookies are 4 byte chunks of memory that are initialized to a random value. These cookies are strategically placed in front of important control structures on the stack, such as saved return addresses. The purpose of security cookies is to detect and prevent certain stack-based buffer overflow attacks that try to overwrite saved return addresses on the stack.

Let’s now review how a malicious user may be able to exploit a stack-based buffer overflow with a standard stack frame that is not protected with security cookies. Then let’s apply the same attack against a stack frame that uses security cookies.

As was previously mentioned, in addition to using security cookies to provide defenses from stack-based buffer overflow attacks, the /GS flag also rearranges the stack frame and adds defensive code to protect vulnerable arguments within a function. To simplify our discussion and better illustrate the use of security cookies, those additional defenses will not be reflected in the stack frames shown here.

(Mouse click)

The vulnerable code that will be exploited is the code shown here. Here is a C language function called *UnsafeFunction* that accepts a char \* named *str*. It allocates 32 bytes on the stack for a buffer called *Buffer*, and then copies the contents of str into Buffer. No validation is made to ensure that the contents of str can fit within the fixed-length boundaries of Buffer; therefore, a stack-based buffer overflow vulnerability exists in this code.

(Mouse click)

The input that will be passed to this function will be the letter “A” repeated forty times. This should be sufficient to trigger the stack-based buffer overflow as the size of the input exceeds the maximum capacity of Buffer.

Let’s now see what happens to a standard stack frame when UnsafeFunction is passed in the letter A repeated forty times.

(Mouse click)

Here’s a standard stack frame that gets created for a call to UnsafeFunction.

(Mouse click)

The 40 “A”s are provided to UnsafeFunction and a buffer overflow occurs. When the function completes and the operating system makes preparations to execute the next instruction, called the function epilogue, the operating system will read the value of the saved return address and try to execute the instruction pointed to by this address.

(Mouse click)

If the input contained malicious code, and the last 4 “A”s where replaced with the address of that malicious code now contained in Buffer, then a malicious user would be able to leverage UnsafeFunction as a conduit to execute arbitrary commands.

### Slide 9 – Defending Return Addresses with Security Cookies

Now let’s see the same scenario as was shown in the previous slide, but now with security cookies protecting the stack.

(Mouse click)

When security cookies are applied, the stack is slightly modified. A 4-byte random value is placed between local variables and control structures. In this example, it is placed between Buffer and the saved frame pointer.

(Mouse click)

The operating system will also save the value of the original security cookie in a different region of memory. Now let’s see what happens when the same forty A’s are provided to UnsafeFunction.

(Mouse click)

Buffer is filled with 32 A’s.

(Mouse click)

The remaining 8 A’s are written over the security cookie and the saved frame pointer.

(Mouse click)

When the function has completed and the function epilogue begins, the operating system will compare the value of the security cookie value on the stack frame against the previously saved value. If the values are different then some buffer corruption has occurred and the operating system will throw an exception. If that exception is not handled then the operating system will stop the application and thus halt the stack-based buffer overflow attack.

### Slide 10 – Buffer Security Check Demonstration

Let’s now see a demonstration of how the protection provided by the /GS flag can be used to prevent the successful exploitation of certain stack-based buffer overflow vulnerabilities.

(Start /GS flag demonstration)

### Slide 11 – Buffer Security Check Limitations

As previously mentioned, none of the compiler defenses provide a ‘silver bullet’ for complete protection against exploitation attempts. With respect to buffer security checks (/GS flag), there are certain situations where this defensive measure may not be applicable or effective.

For example, if a function does not contain a buffer, then buffer security checks are not applied by the compiler. Functions with variable argument lists are also not protected. Also, if a function is marked with the “naked” attribute, which instructs the compiler not to generate code for function setup (called the “function prologue”) or tear down (called the “function epilogue”), then buffer security checks are not applied. These and the other limitations listed here are further evidence as to why compiler defenses should not be used as a replacement for secure coding practices. Defensive measures like these are meant to reduce risk, not eliminate it.

Refer to <http://msdn.microsoft.com/en-us/library/8dbf701c.aspx> for more information.

### Slide 12 – Safe Exception Handling (/SAFESEH)

The next compiler defense that will be discussed is called “safe exception handling.” Whenever an exception occurs in code, it is handled by an exception handler. The address of the exception handler is stored on the stack, which could be overwritten and corrupted or hijacked by a malicious user. If an exception occurs and the operating system tries to execute an exception handler at an address that has been corrupted by a malicious user, that malicious user may be able to control the execution flow of the process. The Code Red worm that exploited the vulnerability addressed by the Microsoft Security Bulletin MS01-033 leveraged this type of vulnerability and could have been prevented if the */SAFESEH* option was available at that time.

When Safe Exception Handling is enabled with the /SAFESEH linker option, the linker writes the addresses of the exception handlers to the portable executable (PE) header. From then on, whenever an exception is raised and the address for the handler for that exception is retrieved from the stack, the operating system will first check the validity of that address before executing the specified exception handler. The operating system will compare the retrieved exception handler address against the valid exception handler addressees stored in the portable executable header. If the retrieved exception handler address from the stack is not contained in the valid list of exception handler addresses, the operating system will halt the process.

### Slide 13 – Hardware-Enforced Data Execution Prevention (/NXCOMPAT)

Hardware-Enforced Data Execution Prevention helps to prevent code from executing in application data pages. Data Execution Prevention has been implemented under several different names: (AMD refers to this as “No eXecute (NX)” and Intel refers to this as “eXecute Disable (XD)”); however, the concept behind Data Execution Prevention remains the same across all such implementations. Data Execution Prevention (DEP) is supported in Windows XP Service Pack 2 and higher.

Hardware-enforced Data Execution Prevention works by marking all non-code memory regions in a running process as non-executable. Under most situations, code should not be executed from non-code memory regions, such as application stacks or heaps. Hardware-enforced Data Execution Prevention will detect attempts to try to execute data and halt the process.

To use this feature, the system processors on which an application is running must support Data Execution Prevention. The operating system must also support this feature. Most modern day processors provide this support and versions of the Windows operating system, XP SP2 and higher, support this feature. Application teams can leverage Data Execution Prevention by using the compilation linker flag */NXCOMPAT*.

Refer to <http://msdn.microsoft.com/en-us/library/ms235442(VS.80).aspx> for more information.

### Slide 14 – Image Randomization

For Windows Vista, Windows Server 2008 and higher, several key operating system images have been compiled using the */DYNAMICBASE* compiler defense. With this compiler defense, the address of those key operating system images in memory are randomized each time they are loaded. The likelihood that any attack that relies on previously known static addresses within those operating system images will be successful is substantially reduced since this aspect of address predictability is eliminated. The feature that enables this randomization is called Address Space Layout Randomization (ASLR). Also, in order to use Address Space Layout Randomization, Data Execution Prevention must be enabled.

Application development teams can apply the same image base address randomization. To do this, the /DYNAMICBASE option needs to be specified when an application is being compiled. The /DYNAMICBASE option is available for Microsoft Visual Studio 2005 SP1 and higher.

Refer to <http://msdn.microsoft.com/en-us/library/bb384887.aspx> for more information.

### Slide 15 – Image Randomization Illustration

Let’s see how image randomization can be used to reduce the likelihood that a malicious user will be able to successfully exploit a buffer overflow vulnerability.

(Mouse click)

Recall that whenever a buffer overflow is exploited, malicious users need to jump or redirect the execution flow to certain known addresses, such as the address of some buffer they have filled with malicious code or a known dynamically linked library (DLL) location.

(Mouse click)

When image randomization is enabled, that known address changes each time the application is loaded. By doing this, malicious users are forced to guess the new address of the code that they are trying to execute. In most cases, malicious users will guess incorrectly and the attack will not succeed. Even the address space of the stack of an application is randomized, which means that correctly jumping into overflowed stack buffers is also difficult.

Again, it is not impossible for a malicious user to correctly determine the correct address of the code they are trying to execute; however, with image randomization, the task of guessing the correct address of the code correctly is much more difficult.

### Slide 16 – MIDL Compiler (/robust)

For applications that implement remote procedure calls (RPCs) and Component Object Model (COM) code, the Microsoft Interface Definition Language (MIDL) */robust* compiler flag can be used to provide additional defenses to applications. When this flag is used, the Network Data Representation (NDR) will perform run-time error checking on correlated arguments in dynamic arrays, unions, and in *out interface pointers*. A correlated argument is an argument that uses attributes that allow the size of the data object to be determined at run-time.

Refer to <http://msdn.microsoft.com/en-us/library/aa367363(VS.85).aspx> for more information.

### Slide 17 – Microsoft SDL Compiler Defenses Windows 32-bit Requirements

For all Windows 32-bit C and C++ applications developed using the Microsoft SDL, the */GS*, */SAFESEH*, */NXCOMPAT* and */robust* compiler flags (when applicable) must be enabled. While /DYNAMICBASE is not currently required by the Microsoft SDL, it is still highly recommended as it can provide an additional layer of defense to applications with very little effort.

In addition to certain compiler defenses that must be enabled, the Microsoft SDL also has requirements on the minimum compiler versions that must be used. Compiler protection has evolved and has been greatly improved upon in later versions of Microsoft compilers. As such, the latest compiler version is always recommended.

For details regarding the compiler defense requirements for other architectures, including minimum compiler version information, please refer to the most current Microsoft SDL guidance paper available at <http://www.microsoft.com/sdl>.

### Slide 18 – Conclusion

This concludes the discussion on Microsoft compiler defenses. Applications compiled with Microsoft compilers can leverage some of the built-in compiler defense mechanisms to improve their resiliency to buffer overflow attacks. These defenses are added to the compiled code automatically by enabling certain flags without requiring any additional effort from developers.

While each of the compiler defenses discussed in this presentation can greatly improve an application’s ability to withstand certain attacks at run-time, these defenses should not be regarded as replacements for writing more secure code and following security best-practices. Each of these compiler defenses reduce the likelihood of successful exploitation but do not entirely eliminate the possibility of successful attacks being employed.

Certain compiler defenses must be enabled for any applications developed following the guidance, tools and processes of the Microsoft SDL. Furthermore, specific compiler versions must also be used in order to ensure that the applied compiler defenses are sufficient to address the current threat landscape.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation which focused on compiler defenses, have been shared with each of you as a way for our organization to enhance our application development practices and the security of our applications.

### Slide 19 - Appendix

This section provides additional slides, materials, and information to supplement the main contents of the presentation.

### Slide 20 – Microsoft Security Development Lifecycle (SDL)

This diagram compares the security engineering steps of the SDL to the software engineering steps of the classic SDLC (software development lifecycle). The blue outer ring represents traditional software development and the orange inner circle represents the SDL. Notice that the security engineering steps are incorporated into the existing software engineering steps and that any engineering task can be supplemented with a security engineering task.

Both of these development lifecycles, or collections of engineering steps, apply to the software development lifecycle regardless of the particular development model you use (for example waterfall, Agile, etc.) The small pewter colored circles represent the various milestones in your model and are an excellent time for ensuring that the steps in both the security and software development lifecycles have been adequately addressed.

The SDL process has been documented and published in *The Security Development Lifecycle* book (Microsoft Press 2006, ISBN: 9780735622142), and the official Web site can be accessed at [http://www.microsoft.com/sdl](http://go.microsoft.com/?linkid=9672761).

### Slide 21 – Microsoft Writing Secure Code Book Series

Microsoft has several publications on secure implementation including the industry leading Writing Secure Code series. Writing Secure Code is mandatory reading for software engineering teams at Microsoft and provides an in-depth discussion of common software weaknesses and effective remedies.

It also provides information with which testers can use to better ensure that the applications they are testing meet security quality assurance requirements.

### Slide 22 – Microsoft Developer Network (MSDN) Security Developer Center

Microsoft also has a security developer center located at [http://msdn.microsoft.com/security](http://go.microsoft.com/?linkid=9672763) where development teams (architects, developers and testers) can find a wealth of resources, including guidance and tools, to help them build safer applications using Microsoft technologies and platforms.

### Slide 23 – Secure Development Blogs

Visit the [SDL Blog](http://go.microsoft.com/?linkid=9672765) to get the most current ideas and thoughts from Microsoft SDL team members.

Visit [Michael Howard’s Blog](http://go.microsoft.com/?linkid=9672764) to read all about how security can be effectively incorporated into the software development process from the author of the popular book, *Writing Secure Code* (Howard, Michael and David LeBlanc, Microsoft Press, Redmond, Washington, 2003).

### Slide 24 – Hunting Security Bugs

Members of the Microsoft Office Security team have written a book that covers common application security issues and how to test for them. More information about this book can be found at [http://www.microsoft.com/mspress/books/8485.aspx](http://go.microsoft.com/?linkid=9672768).

### Slide 25 – Additional SDL Training

Additional SDL training content, such as the following is currently or will be available soon:

**Secure Design Principles:** This content provides application designers with the fundamentals and principles they require to design more secure applications. Other content related to secure design builds upon the knowledge established in this content.

**Secure Implementation Principles:** This content provides developers with the fundamentals and principles they require to develop more secure applications. Other content related to secure implementation builds upon the knowledge established in this content.

**Secure Verification Principles:** This content provides testers and quality assurance personnel with the fundamentals and principles they require to test secure applications. Other content related to secure testing builds upon the knowledge established in this content.

**SQL Injection Vulnerabilities:** SQL injection vulnerabilities are commonly encountered vulnerabilities in applications using a database. As more applications move towards the Web paradigm and are driven by databases, this vulnerability is expected to become even more prolific than is currently being realized. This content provides an overview of SQL injection vulnerabilities and how the SDL can be used to significantly reduce the risk of a SQL injection attack.

**Cross-Site Scripting Vulnerabilities:** Cross-site scripting vulnerabilities are the most commonly encountered Web-based vulnerabilities today. These types of vulnerabilities continue to plague the Web-application world and a user’s ability to trust the applications they are using. This content provides an overview of cross-site scripting vulnerabilities, and how the SDL can be applied to significantly reduce the risk of a cross-site scripting attack.

**Buffer Overflow Vulnerabilities:** Buffer overflows are considered the most dangerous application-level vulnerability. This content provides an overview of buffer overflows, and how the SDL can be used to significantly reduce the risk of a buffer overflow attack.

### Slide 26 – Microsoft Compiler Defenses Resources

Here are additional resources and links on the Microsoft compiler protection features.