MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

CODE ANALYSIS (LEVEL 200)

Guide

Version 1.0

The following documentation provides presenter’s notes for the Microsoft Security Development Lifecycle (SDL) Code Analysis (Level 200) 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](#_Toc233594126)

[1.1 Introduction 2](#_Toc233594127)

[1.2 System Requirements 2](#_Toc233594128)

[1.3 Presentation Themes 2](#_Toc233594129)

[2.0 Microsoft SDL Code Analysis 2](#_Toc233594130)

[Overview 2](#_Toc233594131)

[Presentation Transcript 3](#_Toc233594132)

[Presentation Voiceover 3](#_Toc233594133)

[Presentation Demonstrations 3](#_Toc233594134)

[Slide 2 – Title Slide 3](#_Toc233594135)

[Slide 3 – Agenda 3](#_Toc233594136)

[Slide 4 – Microsoft Security Development Lifecycle (SDL) 3](#_Toc233594137)

[Slide 5 – Code Analysis Overview 4](#_Toc233594138)

[Slide 6 – Static Analysis Versus Binary Analysis 5](#_Toc233594139)

[Slide 7 – Static Source Code Analysis 5](#_Toc233594140)

[Slide 8 – Binary Code Analysis 6](#_Toc233594141)

[Slide 9 – Code Analysis Pros and Cons 6](#_Toc233594142)

[Slide 10 – Microsoft FxCop 7](#_Toc233594143)

[Slide 11 – Microsoft FxCop Demonstration 7](#_Toc233594144)

[Slide 12 – Microsoft PREFast 7](#_Toc233594145)

[Slide 13 – Microsoft PREFast Demonstration 8](#_Toc233594146)

[Slide 14 – Visual Studio Code Analysis 8](#_Toc233594147)

[Slide 15 – Visual Studio Code Analysis Demonstration 8](#_Toc233594148)

[Slide 16 – ASP Source Code Analyzer for SQL Injection 8](#_Toc233594149)

[Slide 17 – ASP Source Code Analyzer for SQL Injection Demonstration 8](#_Toc233594150)

[Slide 18 – Microsoft SDL Code Analysis Requirements 8](#_Toc233594151)

[Slide 19 – Conclusion 9](#_Toc233594152)

[Slide 20 - Appendix 9](#_Toc233594153)

[Slide 21 – Microsoft Security Development Lifecycle (SDL) 9](#_Toc233594154)

[Slide 22 – Microsoft Writing Secure Code Book Series 9](#_Toc233594155)

[Slide 23 – Microsoft Developer Network (MSDN) Security Developer Center 9](#_Toc233594156)

[Slide 24 – Secure Development Blogs 10](#_Toc233594157)

[Slide 25 – Hunting Security Bugs 10](#_Toc233594158)

[Slide 26 – Additional SDL Training 10](#_Toc233594159)

# 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:

* Code Analysis (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 Code Analysis

## Overview

Modern day applications are becoming more complex and more feature-rich. As a result, application code sizes have also increased along with the possible areas of attack exposure. Organizations practicing security assessment best practices, such as conducting ongoing code reviews, require ways with which to help scale and socialize such best practices. Further, organizations practicing these security assessment best practices must balance such activities with those activities required to minimize applications’ exposure to attack. Code analysis tools offer one possible way this objective may be achieved. This presentation provides an overview of code analysis tools from Microsoft. This presentation also describes how these code analysis tools may be used to reduce overall application exposure to attack through the Microsoft SDL.

The insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on code analysis, 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 Code Analysis (Level 200) presentation. The precise transcript text provided herein is also incorporated into the notes section of each slide in the Microsoft PowerPoint Code Analysis Feature (Level 200) presentation itself for ease of reference.

## Presentation Voiceover

A voiceover of the Code Analysis (Level 200) 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 Code Analysis (Level 200) 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: Code Analysis](http://go.microsoft.com/?linkid=9672754)

### Slide 2 – Title Slide

The Code Analysis (Level 200) presentation introduces the role that the Microsoft Security Development Lifecycle (SDL) fulfills in trusted application development. It also provides an overview of several Microsoft code analysis tools that application development teams can use to analyze their applications and reduce their overall attack exposure. This presentation also provides an overview of how code analysis is used within the Microsoft SDL to help Microsoft deliver safer and more trusted applications to its customers.

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

### Slide 3 – Agenda

In this presentation, we will complete an overview of the Microsoft SDL, code analysis, several Microsoft code analysis tools, and how Microsoft uses code analysis through its SDL process to deliver safer and more trusted applications to its customers.

The specific Microsoft code analysis tools that will be discussed in this presentation are FxCop, PREFast, the Visual Studio Code Analysis feature, and ASP Source Code Analyzer for SQL Injection.

### 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 vulnerabilities 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 SDL process.

As was previously indicated, this presentation focuses on code analysis, and available Microsoft tools provided through the Microsoft SDL.

### Slide 5 – Code Analysis Overview

Code analysis tools are software tools that analyze application implementations to help ensure that development best practices are followed and highlight possible areas of improvement when deviations from development best practices are encountered. When implementation best practices are not followed, security vulnerabilities, such as buffer overflows and SQL injection, manifest in code and can be exploited by malicious users. Therefore, the Microsoft SDL stresses, and in certain cases requires the use of code analysis tools to better ensure that applications built with the Microsoft SDL are more safe and trustworthy.

The two types of code analysis tools that will be discussed in this presentation are static source code analysis tools and binary analysis tools. Both types of code analysis tools share the common goal of analyzing application implementations for areas of improvement, however, they differ only in how they arrive at this goal.

It should be noted that while code analysis tools can be useful in aiding developers and testers to identify security vulnerabilities in their application implementations, they should not be used as a replacement for implementation and verification best practices. Code analysis tools will be able to identify certain, but not all coding practices that can lead to exploitable vulnerabilities. Building more secure applications requires several steps, and the use of code analysis tools is just one of a series of several steps.

The use of code analysis tools is particularly stressed within the Microsoft SDL Implementation and Verification phases. Within each of these phases, developers and testers, respectively, can detect software issues early within the software development lifecycle and apply code corrections when the cost of doing so is minimal. It is Microsoft’s experience that this is preferable to the alternative and more common scenario where code issues are addressed after the application has been fully developed and delivered to customers. This is due to the cost of re-engineering the application being much higher than if issues were encountered and addressed early in the development process.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on code analysis, 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 – Static Analysis Versus Binary Analysis

This slide provides a visual illustration of the fundamental differences between static code analyzers and binary code analyzers. Let’s take a look at the application compilation process and then see where static and binary code analysis is applied.

(Mouse click)

Whenever an application is implemented, it is first implemented in 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, which a linker then takes and transforms into a binary file, such as a library or executable.

(Mouse click)

Let’s first discuss static source code analysis and see where within the compilation process it is applied.

(Mouse click)

Static source code analysis examines the actual source code written by developers for exploitable coding patterns. Any potentially vulnerable code detected is flagged and reported to the user of the tool. Binary analysis tools also read the source implementation of an application; however, binary analysis tools read the source implementation of an application in a different form. Specifically, binary analysis tools read the binary or machine code form of an application and then analyze it for vulnerable coding patterns.

(Mouse click)

Let’s now discuss some of the available Microsoft code analysis tools that application teams can use to analyze and reduce their application’s exposure to attack.

### Slide 7 – Static Source Code Analysis

Static source code analysis tools are software applications that analyze the uncompiled source code of an application. Example inputs to these types of tools might be C, C++, and C# source code files. Outputs are a listing of areas within the source code, such as line numbers and function names, where the code may be exploitable by malicious users.

As the name implies, static source code analysis tools are useful only when the source code of an application is available. An advantage of these types of tools is that since the input and output from these tools are based on human readable source code, developers can easily locate and diagnose coding issues. Another advantage of static source code analysis tools is that when compared to binary analysis, static source code analysis is the more mature code analysis technology. Static source code analysis has been researched and developed longer, and leverages advances in areas, such as data flow analysis, model checking, and code annotation to improve results.

### Slide 8 – Binary Code Analysis

Binary code analysis tools, similar to static source code analysis tools, analyze the implementation of an application. However, instead of analyzing the *uncompiled* source code of an application, they examine the *compiled* or *binary* version of the application. You may hear the binary version of an application sometimes being referred to as *machine code*.

Binary analysis tools have a distinct advantage over their static source code counterparts in that the inputs into these tools are the actual applications. Source code that is read by static source code analysis tools may be modified or rearranged by optimizing compilers. This final and actual state of the application often will not be visible to static source analysis tools, but will be visible to binary analysis tools. The advantage of this is that reported findings tend to be more accurate.

This higher degree of accuracy, however, comes at a cost. Reported findings are based on the machine code that binary analysis tools read. As a result, it may be difficult for developers to trace those machine code findings back to actual human readable source code.

### Slide 9 – Code Analysis Pros and Cons

Whether you are using static source code analysis or binary analysis, code analysis tools have their benefits and disadvantages.

Let’s first start with the pros.

(Mouse click)

The first major benefit of code analysis tools is that they can be used to help scale the code review process by reducing the manually intensive aspects of the code review process. Reviewing code entails inspecting static source code, line by line, for common coding patterns that can be exploited. Security testers often manually perform code review as a means of assessing an application’s security posture. Since modern day applications can exceed well over hundreds of thousands of lines of code, code analysis tools can greatly help when analyzing large code bases.

(Mouse click)

The next major benefit of code analysis tools is that they can help assess compliance to secure-coding policies. If your organization has defined secure-coding policies, such as policies that dictate which libraries are prohibited for use (as is prescribed within the Microsoft SDL), then code analysis tools can be a highly efficient way of determining if those policies have been satisfied accordingly.

Let’s now look at the cons.

(Mouse click)

The first major disadvantage of code analysis tools is that they can often produce false positives. False positives are detected code issues, which are in fact not issues at all. Too many false positives will frustrate developers, as they will be spending large amounts of time chasing non-issues. With enough false positives, developers eventually will stop using the tool altogether.

(Mouse click)

The opposite of false positives is false negatives. False negatives are actual issues in code that the code analysis tool has failed to detect. The danger of false negatives is that they leave development teams with the false impression that their application is “secure” when in fact it may not be.

(Mouse click)

The next disadvantage of code analysis tools is that they tend to be language-specific. One tool may analyze only unmanaged code but not managed code. If your application uses a mixture of languages for its implementation, then using several code analysis tools may be required.

(Mouse click)

The final prevalent disadvantage of code analysis tools is that they may not be able to find design-level issues. Since the inputs into these tools are typically binary or static source code, the issues they can report are generally issues based on that source code. Issues related to the design of the application that the source code implements may not be detected.

### Slide 10 – Microsoft FxCop

The first code analysis tool that will be discussed in this presentation is Microsoft’s FxCop analysis tool. FxCop is a binary code analysis tool that examines .NET Framework assemblies for conformance to the Microsoft .NET Framework Design Guidelines. The FxCop tool homepage can be found at <http://msdn.microsoft.com/en-us/library/bb429476(vs.80).aspx>. Information regarding the Microsoft.NET Framework Design Guidelines can be found at <http://msdn.microsoft.com/en-us/library/czefa0ke.aspx>.

FxCop can be used as a graphical user interface tool, as well as a command line tool. This allows development teams to fully integrate FxCop into the software development lifecycle.

In addition to security checks, FxCop also analyzes assemblies for conformance to certain design, localization, and performance best practices.

Please remember that FxCop, and the other tools you will see in this presentation, are meant to help developers and testers improve their code. They will not catch all possible vulnerabilities. In fact, in some of the later tool demonstrations, certain vulnerabilities that are not detectable by tools but are detectable by other means, such as through performing a security code review, will be highlighted.

### Slide 11 – Microsoft FxCop Demonstration

Let’s see a demonstration of FxCop.

(Start FxCop demonstration)

### Slide 12 – Microsoft PREFast

Microsoft PREFast is a static source code analysis tool that identifies vulnerabilities in C/C++ code. In contrast to the FxCop tool, PREFast reads the source code implementation of applications rather than the compiled binaries. Like FxCop, this tool is freely available as a download, and information regarding downloading PREFast can be found on MSDN at <http://www.microsoft.com/whdc/DevTools/tools/PREfast.mspx>.

Also like FxCop, PREFast can be used as a graphical user interface tool as well as a command line tool. This gives PREFast the ability to be fully integrate into the software development lifecycle. Finally, PREFast is distributed with the Windows Driver Kit (WDK); however it can be used to analyze non-driver code written in C/C++.

### Slide 13 – Microsoft PREFast Demonstration

Let’s see a demonstration of PREFast.

(Start PREFast demonstration)

### Slide 14 – Visual Studio Code Analysis

If organizations prefer to access the features and capabilities of PREFast and FxCop in a more integrated fashion, rather than as standalone tools, such organizations can do so by using Visual Studio Team Edition and higher versions. Information about Visual Studio can be found at <http://www.microsoft.com/visualstudio>.

The code analysis features of Visual Studio can be enabled using the /analyze compiler switch for command-line application build scenarios or via the *project properties* settings for applications built within the Visual Studio integrated development environment (IDE).

### Slide 15 – Visual Studio Code Analysis Demonstration

Let’s see a demonstration of the Microsoft Visual Studio /analyze feature.

(Start Visual Studio Code Analysis demonstration)

### Slide 16 – ASP Source Code Analyzer for SQL Injection

In June 2008, Microsoft released a static source code analysis tool that helps developers and testers find SQL injection vulnerabilities in Active Server Page (ASP) code.

This tool is available as a command-line static source code analysis tool only, and can only analyze ASP pages that are written in VBScript. Refer to <http://support.microsoft.com/kb/954476> for more information regarding this tool.

### Slide 17 – ASP Source Code Analyzer for SQL Injection Demonstration

Let’s see a demonstration of the ASP Source Code Analyzer for SQL Injection.

(Start Source Code Analyzer for SQL Injection in ASP code demonstration)

### Slide 18 – Microsoft SDL Code Analysis Requirements

Organizations that currently use code analysis tools or are considering incorporating code analysis tools into their software development lifecycles should consider instituting policies that state clearly which code analysis tool warnings or findings are acceptable and those that are not. By specifying code analysis tool requirements within policy, confusion or disagreements that may arise in response to discovered code issues may be reduced accordingly. For example, one developer may feel that a certain type of vulnerability has a high criticality rating whereas another developer may not. Creating code analysis tool requirements through policy can help eliminate these types of potential discrepancies. Code analysis tools can also often generate large amounts of false positives. Therefore, by specifying clear and concise code analysis tool policies, the developers can focus their efforts on debugging and diagnosing issues that will actually require their attention vs. having to concentrate on *all* issues.

The Microsoft SDL documents the exact PREFast, FxCop, and Visual Studio code analysis feature warnings that Microsoft developers must address prior to releasing an application to customers. For instance, all unmanaged code must pass PREFast analysis and must not contain issues and warnings, such as 4532, 6029, and so on. Any application that is developed within Microsoft using the Microsoft SDL must adhere to these requirements. Organizations can also adopt these same proven code analysis tool requirements within their own organizations. More information regarding the code analysis tool requirements of the Microsoft SDL can be found within the [Microsoft SDL whitepaper](http://go.microsoft.com/?linkid=9672762) or in chapter 21 of the Microsoft SDL book (refer to [http://www.microsoft.com/mspress/books/8753.aspx](http://go.microsoft.com/?linkid=9672766))**.**

### Slide 19 – Conclusion

This concludes the discussion on code analysis. Code analysis tools analyze the implementation of an application for areas of improvement and coding patterns that can be exploited by a malicious user. Code analysis tools can be categorized as static source analyzers or binary analyzers. Both types of code analysis tools can be effective in helping development teams identify application implementation issues; however, neither represents a silver bullet for developing safer software. Code analysis tools should be used in addition to, and to augment safer development practices, rather than as a replacement.

Microsoft provides several standalone code analysis tools, such as PREFast and FxCop, for developers using Microsoft’s platforms and technologies. An integrated code analysis environment is available to developers using Visual Studio Team Edition and higher versions. Microsoft also provides ASP developers with a standalone command-line code analysis tool to help identify certain SQL injection vulnerabilities.

To use code analysis tools effectively, organizations should develop specific code analysis tool requirements within their software development lifecycle policies. The Microsoft SDL provides developers with specific requirements for PREFast, FxCop, and the Visual Studio /analyze feature to help developers create safer and more trusted applications. Organizations that wish to base their code analysis requirements on proven processes and policies can do so by adopting those defined within the Microsoft SDL.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation which focused on code analysis, 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 20 - Appendix

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

### Slide 21 – 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 22 – 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 23 – 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 24 – 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 25 – 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 26 – 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.