MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

SQL INJECTION (LEVEL 200)

Guide

Version 1.0

The following documentation provides presenter’s notes for the Microsoft Security Development Lifecycle (SDL) SQL Injection Vulnerabilities (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](#_Toc233631080)

[1.1 Introduction 2](#_Toc233631081)

[1.2 System Requirements 2](#_Toc233631082)

[1.3 Presentation Themes 2](#_Toc233631083)

[2.0 SDL SQL Injection Vulnerabilities 2](#_Toc233631084)

[Overview 2](#_Toc233631085)

[Presentation Transcript 2](#_Toc233631086)

[Presentation Voiceover 2](#_Toc233631087)

[Presentation Demonstrations 3](#_Toc233631088)

[Slide 2 – Title Slide 3](#_Toc233631089)

[Slide 3 – Agenda 3](#_Toc233631090)

[Slide 4 – Microsoft Security Development Lifecycle (SDL) 3](#_Toc233631091)

[Slide 5 – SQL Injection Overview 4](#_Toc233631092)

[Slide 6 – SQL Injection Example 4](#_Toc233631093)

[Slide 7 – SQL Injection in the News 5](#_Toc233631094)

[Slide 8 – SQL Injection Demonstration: The Contoso Credit Union 5](#_Toc233631095)

[Slide 9 – Common SQL Injection Myths 6](#_Toc233631096)

[Slide 10 – Preventing SQL Injection with the Microsoft SDL 6](#_Toc233631097)

[Slide 11 – Using Parameterized Queries to Prevent SQL Injection Attacks 7](#_Toc233631098)

[Slide 12 – SQL Injection Scanning Tools 8](#_Toc233631099)

[Slide 13 – Conclusion 8](#_Toc233631100)

[Slide 14 - Appendix 8](#_Toc233631101)

[Slide 15 – Microsoft Security Development Lifecycle (SDL) 8](#_Toc233631102)

[Slide 16 – Microsoft Writing Secure Code Book Series 9](#_Toc233631103)

[Slide 17 – Microsoft Developer Network (MSDN) Security Developer Center 9](#_Toc233631104)

[Slide 18 – Secure Development Blogs 9](#_Toc233631105)

[Slide 19 – Hunting Security Bugs 9](#_Toc233631106)

[Slide 20 – Additional SDL Training 9](#_Toc233631107)

[3.0 Additional SQL Injection Information Resources 10](#_Toc233631108)

# 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:

* Microsoft SDL Training – SQL Injection Vulnerabilities (Level 200)

## 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 SDL SQL Injection Vulnerabilities

## Overview

For many years, malicious users have been exploiting database-driven applications using a vulnerability called SQL injection. This presentation provides an overview of SQL injection vulnerabilities along with a discussion on how the Microsoft SDL can be applied to prevent attacks based on this type of vulnerability.

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

## Presentation Voiceover

A voiceover of the SQL Injection Vulnerabilities (Level 200) presentation transcript below, approximately 35 minutes in length, is also available to assist the presenter in becoming sufficiently acclimated with the subject matter addressed in the SQL Injection Vulnerabilities (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: SQL Injection Vulnerabilities](http://go.microsoft.com/?linkid=9672760)

### Slide 2 – Title Slide

The SQL Injection Vulnerabilities (Level 200) presentation introduces the role that the Microsoft Security Development Lifecycle (SDL) fulfills in trusted application development. It also provides an overview of one of the most common vulnerabilities encountered today called SQL injection, as well as a discussion on how the Microsoft SDL can be applied to prevent attacks based on the exploitation of this vulnerability.

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, SQL injection vulnerabilities, and how attacks based on this particular type of vulnerability can be prevented through successfully employing the Microsoft SDL.

### 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 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 SQL injection vulnerabilities, and how they can be prevented using the guidance, process and tools provided by the SDL.

### Slide 5 – SQL Injection Overview

SQL injection vulnerabilities are one of the most commonly encountered application security vulnerabilities today. SQL injection can occur whenever unvalidated data is used to construct dynamic SQL statements that are later executed by a database. This type of vulnerability can give malicious users the ability to control the execution flow of the affected SQL statements. With this ability, malicious users can execute unauthorized commands using the database privileges granted to your application. The potential for damage increases especially in scenarios where your application is granted excessive or highly-privileged database connection rights.

Like many other application security implementation issues, SQL injection vulnerabilities stem from the lack of or improper input validation. This situation is further exacerbated if the input data is used in the construction of dynamic SQL statements without the use of type-safe parameters, or if the connection used to access the database has excessive rights.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on SQL Injection, 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 – SQL Injection Example

Before you see a demonstration of a SQL injection attack, let’s take a few moments to establish an understanding of the internals of SQL injection.

Here is a sample SQL statement that reads a tracking ID and selects all relevant data to that ID from the ShipmentOrders database table. This scenario is typical in applications where a user has purchased an item and needs to be able to track its delivery en route to the final destination. Let’s now look at different input scenarios, the resulting SQL statement that gets executed, and how each can lead to exploitable conditions.

(Mouse click)

In the first scenario, input ID is set to 1000. The database simply executes SELECT \* FROM ShipmentOrders WHERE ID=‘1000’; and retrieves the data for the order corresponding to ID 1000, if it exists. This is the expected and correct behavior. Now let’s look at some more interesting ID inputs and see how they can be used by malicious users to perform unauthorized actions.

(Mouse click)

In the second scenario, ID is specified by the malicious user as 1000, but some other data is also included. Notice here the malicious user is entering the shipment ID as 1000’; DROP TABLE ShipmentOrders;--. When this gets inserted into the original SELECT statement above, the database actually sees two different SQL queries to execute. The first is to retrieve the information for the order corresponding to ID 1000, and the second is to drop the table labeled ShipmentOrders. If this were a real-life situation, all shipment data stored in that table would be deleted. This is very bad!

(Mouse click)

In the third and last scenario, ID is specified by the malicious user as 1000’; exec xp\_cmdshell and then some command to execute it. As with the previous scenario, the database server sees two different SQL queries to execute. The first is to retrieve information for the order corresponding to ID 1000 and the second is to execute the xp\_cmdshell stored procedure and have it execute any command that the malicious user provides. This is horrible!

***Note:*** Double dashes (--) is the SQL equivalent of comment characters in a programming language like C#, C or C++. It indicates to the database that anything after the double dashes should be considered as a comment and not executed. In the attacks above, the additional single (‘) quote that the malicious user inserted would have imbalanced the number of quotes in the executing SQL query. The double dashes was used to comment out the final quote and semi-colon that would have imbalanced the SQL query, caused a database error and prevented the attack from succeeding.

### Slide 7 – SQL Injection in the News

SQL injection attacks have been well understood for many years, yet they are still one of the most commonly encountered vulnerabilities today. In 2008, SQL injection ranked as the second most prevalent Web application security vulnerability (<http://www.owasp.org/index.php/Top_10_2007>). Here are, for example, some news stories regarding SQL injection.

In March 2006, the tourism website of the Indian government was found to be vulnerable to SQL injection attacks. A story about this report can be found at <http://www.webappsec.org/projects/whid/list_id_2006-27.shtml>.

In August 2007, the website of the United Nations was defaced using a SQL injection attack. The main Web site’s content was replaced with messages regarding worldwide political issues. A story about this attack can be found at <http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9030318>.

More recently, in January 2008 a mass SQL injection attack occurred, resulting in tens of thousands of systems being attacked. Victims of the mass attack included numerous government, academic and private sector entities. A story about this attack can be found at <http://www.computerworld.com.au/index.php/id;683627551>.

### Slide 8 – SQL Injection Demonstration: The Contoso Credit Union

Now it is time to take a look at some actual SQL injection vulnerabilities, and see how a malicious user can exploit these vulnerabilities. In the coming demonstration, we will be taking on the role of the malicious user and attacking a demonstration Web site called the Contoso Credit Union (CCU).

As indicated in the previous slide, please note that while the demonstration you are about to see is a series of SQL injection attacks against a Web-based application, SQL injection attacks are not limited to Web-based applications. Any application, regardless of the platform or language it has been implemented with that uses a database, regardless of the database application, is potentially susceptible to this type of attack.

Let’s begin the demonstration.

(Start SDL Demonstration Platform Contoso Credit Union SQL injection attack demonstration)

### Slide 9 – Common SQL Injection Myths

There are several common myths regarding SQL injection vulnerabilities and their exploitability. In this slide we will look at some of these common myths and discuss why each is inaccurate.

(Mouse click)

**Myth #1: SQL injection applies only to applications that use Microsoft SQL Server:** One of the most common myths about SQL injection is that it is a problem that is exclusive to applications built on Microsoft SQL Server. This simply is not true. SQL injection utilizes the SQL language, which is a standard database language designed to retrieve and manage data stored on relational databases. Any database that understands SQL is potentially susceptible to this type of attack. While the SQL injection demonstration you just saw did use Microsoft SQL Server, remember that any database that processes SQL, such as Microsoft SQL Server, Oracle, and MySQL, is susceptible to similar attacks.

(Mouse click)

**Myth #2: SQL injection applies only to Web-based applications:** The second common myth regarding SQL injection is that SQL injection vulnerabilities are only applicable in Web-based application scenarios. This again is simply not true. Certainly, SQL injection is encountered quite frequently in Web-based scenarios; however, applications that do not use a Web-based front end, such as traditional client-server applications, are equally as susceptible. Remember, any application that uses a database in some manner is potentially susceptible to SQL injection attacks.

(Mouse click)

**Myth #3: Transport SQL injection can be remedied by using transport security protocols, such as SSL and IPSec:** Another common myth regarding SQL injection is that transport security protocols, such as secure socket layers (SSL) and IP security (IPSec), can provide protection from SQL injection attacks. The SQL language is processed at the application-level. Therefore, any attack based on SQL also exists at the application-level. Protocols, such as SSL and IPSec, are processed prior to the application-level; they are processed at the network-level. Therefore, the use of these protocols does not affect a malicious user’s ability to successfully execute a SQL injection attack.

(Mouse click)

**Myth #4: SQL SQL injection can be remedied by using transport security protocols, such as SSL and IPSec:** The final myth regarding SQL injection is that SQL injection attacks can only be conducted by non-authenticated malicious users; that is, the threat of SQL injection attacks from authenticated users is not possible. This myth incorrectly assumes that database queries will not be processed based on input data provided by non-authenticated users. The reality is that database queries are dynamically constructed and executed on behalf of both unauthenticated users and authenticated users. Therefore, the threat of SQL injection attacks exists for both types of users.

### Slide 10 – Preventing SQL Injection with the Microsoft SDL

Recall that SQL injection is an application vulnerability that manifests due to developers not properly validating data that is used to build dynamic SQL statements. Properly validating data for correctness can greatly reduce the threat of SQL injection attacks. Input validation is a broad stroke preventive measure that all developers should be practicing to reduce the number of application vulnerabilities, and not just SQL injection. Specific to SQL injection, the SDL has three requirements.

**SQL Parameterized Queries:** The first SDL requirement is that any application that accesses a database must do so using parameterized queries. Parameterized queries use *placeholders*, or sometimes referred to as *parameterized commands*, to specify input data. Using parameterized queries allows the database executing the query to distinguish between executable and non-executable SQL data. In a sense, with parameterized commands you are indicating to the database that this section of the SQL statement is data only, and must not be executed should it contain valid SQL commands. A demonstration regarding how parameterized queries can be used to remediate the SQL injection attack successfully executed against Contoso Credit Union will be shown shortly.

**Stored Procedures:** Another requirement of the SDL concerning SQL injection is that any application accessing a database should do so using only stored procedures, and that the stored procedures must not use the exec @sql construct. Using stored procedures helps reduce the threat of SQL injection attacks since type-checking is available for parameters. Any attempt to violate the specified type, as in the case with most SQL injection attacks, results in an application exception.

Using the exec @sql construct essentially reproduces the same SQL injection conditions now at the stored procedure level rather than at the application code level.

Note that stored procedures do not remediate SQL injection vulnerabilities. Rather, they make it more difficult for malicious users to successfully conduct SQL injection attacks.

**SQL Execute-only Permission:** The final SDL requirement regarding SQL injection is to “only grant execute permissions on all stored procedures, and grant that permission only for the application domain group” (http://blogs.msdn.com/sdl/archive/2008/05/15/giving-sql-injection-the-respect-it-deserves.aspx). With this defensive measure, if a malicious user attempts to access any database object outside the scope of the stored procedure they are attacking, the database will greatly limit if not entirely block that requested access. In a sense, this defensive measure concedes that a malicious user will be able to exploit some SQL statement in some way; however the potential damage that can be inflicted by that malicious user will be isolated and contained.

### Slide 11 – Using Parameterized Queries to Prevent SQL Injection Attacks

Let’s now take a look at the actual Contoso Credit Union code that was exploited during the demonstration, and see how to use parameterized queries to prevent future attacks. During the demonstration there were in fact two SQL statements that were exploited, and this slide will focus only on the first. The techniques that will be shown here can be applied to both SQL statements.

(Mouse click)

The first exploitable SQL statement is shown here. This was the SQL statement that was used to ensure that each entry had a unique email address associated with it. In this code snippet, the email address is concatenated with the SELECT statement, which is later passed to the SQL command object for eventual execution. It was due to this concatenation that malicious users were able to conduct a SQL injection attack. Here is how similar code would appear using SQL parameterized queries to prevent SQL injection attacks.

(Mouse click)

When refactoring existing SQL statements to use parameterized queries, the first task is to remove any concatenation or insertion of input data and replace it with place holders.

(Mouse click)

If you focus your attention on the first highlighted section in the second code box, the concatenation of the email text box has been removed, and has been replaced with “@pEmail”, which will serve as our SQL parameter place holder.

(Mouse click)

Next, a SQL parameter object is created and set to look for the place holder called “@pEmail”. The parameter is then assigned the value from the email text box. When the database executes the SQL statement, it will look for the “@pEmail” placeholder and substitute the data-only value contained in this SQL parameter object.

(Mouse click)

The last step is to add that parameter to the SQL command as shown here. With three simple code changes, this section of our application is now resistant to SQL injection attacks. Let’s apply this implementation technique to the Contoso Credit Union and see how parameterized queries prevent future SQL injection attacks.

(Start SDL Demonstration Platform Contoso Credit Union SQL injection prevention demonstration)

### Slide 12 – SQL Injection Scanning Tools

In addition to safer coding practices to prevent SQL injection, you should also be regularly reviewing your code for this type of vulnerability. Any section of code that is accessing a database with a dynamically built SQL statement is suspect. Microsoft has published several code scanning tools to assist developers and testers in this endeavor. It should be noted that these tools are designed to assist in code review, and not to replace / obviate the overall code review process. These tools are effective at identifying only certain coding patterns that can lead to SQL injection and nothing else; these tools should be combined with expert manual code review and other security verification practices.

**Microsoft FxCop:** Microsoft FxCop is a binary analysis tool that examines compiled .NET Framework assemblies for common coding weaknesses. It provides checks for simple SQL injection vulnerabilities. Microsoft FxCop can be downloaded at <http://code.msdn.microsoft.com/Release/ProjectReleases.aspx?ProjectName=codeanalysis&ReleaseId=553>.

**Microsoft Visual Studio Code Analysis Feature:** Microsoft has integrated the abilities of FxCop into certain versions of Microsoft Visual Studio (Team Edition and higher). Developers and testers can use this feature through the “/analyze” switch to help identify certain SQL injection vulnerabilities and other common security coding weaknesses in their application implementations. More information regarding this feature is located at <http://msdn.microsoft.com/en-us/library/ms173498.aspx>.

**Microsoft Source Code Analyzer for SQL Injection:** Microsoft released a tool to help developers identify certain SQL injection weaknesses in their ASP code. The tool can be downloaded from <http://www.microsoft.com/downloads/details.aspx?FamilyID=58a7c46e-a599-4fcb-9ab4-a4334146b6ba&displaylang=en>.

### Slide 13 – Conclusion

This concludes the discussion on SQL injection vulnerabilities. In this presentation, an overview of SQL injection vulnerabilities was provided, along with a demonstration illustrating the potential damage that may be inflicted from successfully executed SQL injection attacks. The presentation then concluded by providing an overview of how the Microsoft SDL can be applied to prevent SQL injection attacks.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation which focused on SQL Injection, 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 14 - Appendix

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

### Slide 15 – 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 16 – 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 17 – 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 18 – 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 19 – 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 20 – 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.

# 3.0 Additional SQL Injection Information Resources

Here is a listing of additional resources for more information about SQL injection vulnerabilities:

* Neilcar's blog on recent SQL Injection attacks. <http://blogs.technet.com/neilcar/archive/2008/03/14/anatomy-of-a-sql-injection-incident.aspx>
* SQLInjectionFinder.exe: Determine what ASP Pages were used to carry out recent SQL Injections. <http://www.codeplex.com/Release/ProjectReleases.aspx?ProjectName=WSUS&ReleaseId=13436>
* How To: Protect From SQL Injection in ASP.NET. <http://msdn.microsoft.com/en-us/library/ms998271.aspx>
* Building Secure ASP.NET Applications - Authentication, Authorization, and Secure Communication. <http://msdn.microsoft.com/en-us/library/aa302415.aspx>
* Improving Web Application Security - Threats and Countermeasures. <http://www.microsoft.com/downloads/details.aspx?FamilyId=E9C4BFAA-AF88-4AA5-88D4-0DEA898C31B9&displaylang=en>
* 10 Steps to Help Secure SQL Server 2000. <http://labs.microsoft.com/sql/prodinfo/previousversions/securingsqlserver.mspx>
* Security Guidance for SQL Server: <http://www.microsoft.com/technet/security/prodtech/SQLServer.mspx>
* URL Scan (IIS Security tool that can be used to help prevent these types of attacks). <http://www.microsoft.com/technet/security/tools/urlscan.mspx>
* KB article that details how to configure URL Scan. <http://support.microsoft.com/kb/326444/EN-US>
* Secure Windows Initiative (SWI) article on SQL injection attacks: <http://blogs.technet.com/swi/archive/2008/05/29/sql-injection-attack.aspx>
* Preventing SQL injections in ASP. <http://msdn.microsoft.com/en-us/library/cc676512.aspx>
* UrlScan v3.1 to help filter against SQL injection attempts. <http://www.iis.net/go/1697> (x86) and <http://www.iis.net/go/1698> (x64)
* IIS UrlScan Tutorials:
* <http://learn.iis.net/page.aspx/473/using-urlscan/>
* <http://learn.iis.net/page.aspx/475/urlscan-setup/>
* <http://learn.iis.net/page.aspx/476/common-urlscan-scenarios/>
* <http://learn.iis.net/page.aspx/477/urlscan-faq/>
* SQL Injection Blog Postings:
* <http://blogs.iis.net/wadeh/archive/2008/10/31/urlscan-3-1.aspx>
* <http://blogs.technet.com/neilcar/archive/2008/10/31/sql-injection-hijinks.aspx>

**3rd Party Tools and Information**

Here are links to 3rd party tools and information regarding SQL injection vulnerabilities:

* ISAPI wildcard filter for IIS6. <http://www.codeplex.com/IIS6SQLInjection>
* Web tool to convert Hex to Text. <http://www.paulschou.com/tools/xlate>
* Nazim's IIS Security Blog - Filtering SQL injection from Classic ASP. <http://blogs.iis.net/nazim/archive/2008/04/28/filtering-sql-injection-from-classic-asp.aspx>
* Scan for SQL Injection weaknesses. <http://www.parosproxy.org/download.shtml>
* Advanced SQL Injection in SQL Server Applications. <http://www.ngssoftware.com/papers/advanced_sql_injection.pdf>
* More Advanced SQL Injection. <http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf>