MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

CROSS-SITE SCRIPTING (LEVEL 200)

Guide

Version 1.0

The following documentation provides presenter’s notes for the Microsoft Security Development Lifecycle (SDL) Cross-Site Scripting (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](#_Toc233604988)

[1.1 Introduction 2](#_Toc233604989)

[1.2 System Requirements 2](#_Toc233604990)

[1.3 Presentation Themes 2](#_Toc233604991)

[2.0 SDL Cross-Site Scripting Vulnerabilities 2](#_Toc233604992)

[Overview 2](#_Toc233604993)

[Presentation Transcript 2](#_Toc233604994)

[Presentation Voiceover 2](#_Toc233604995)

[Presentation Demonstrations 3](#_Toc233604996)

[Slide 2 – Title Slide 3](#_Toc233604997)

[Slide 3 – Agenda 3](#_Toc233604998)

[Slide 4 – Microsoft Security Development Lifecycle (SDL) 3](#_Toc233604999)

[Slide 5 – Cross-Site Scripting Overview 4](#_Toc233605000)

[Slide 6 – Cross-Site Scripting In The News 4](#_Toc233605001)

[Slide 7 – Types of Cross-Site Scripting 5](#_Toc233605002)

[Slide 8 – Type 1: Non-Persistent Cross-Site Scripting 5](#_Toc233605003)

[Slide 9 – Type 2: Persistent Cross-Site Scripting 6](#_Toc233605004)

[Slide 10 – Cross-Site Scripting Demo: The Contoso Credit Union 7](#_Toc233605005)

[Slide 11 – Common Cross-Site Scripting Myths 7](#_Toc233605006)

[Slide 12-13 – Reducing the Exposure to Cross-Site Scripting Attacks 8](#_Toc233605007)

[Slide 14 – How Encoding Works 9](#_Toc233605008)

[Slide 15 – Encoding Example 10](#_Toc233605009)

[Slide 16 – Reducing Exposure to Cross-Site Scripting Attacks Demonstration 10](#_Toc233605010)

[Slide 17 – Cross-Site Scripting Scanning Tools 10](#_Toc233605011)

[Slide 18 – Conclusion 11](#_Toc233605012)

[Slide 19 - Appendix 11](#_Toc233605013)

[Slide 20 – Microsoft Security Development Lifecycle (SDL) 11](#_Toc233605014)

[Slide 21 – Microsoft Writing Secure Code Book Series 11](#_Toc233605015)

[Slide 22 – Microsoft Developer Network (MSDN) Security Developer Center 11](#_Toc233605016)

[Slide 23 – Secure Development Blogs 11](#_Toc233605017)

[Slide 24 – Hunting Security Bugs 12](#_Toc233605018)

[Slide 25 – Additional SDL Training 12](#_Toc233605019)

[Slide 26 – Reducing Exposure to Cross-Site Scripting Attacks using ASP.NET 12](#_Toc233605020)

# 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 – Cross-Site Scripting 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 Cross-Site Scripting Vulnerabilities

## Overview

An increasing number of organizations are transitioning their line of business applications towards the Web platform to make use of new market opportunities and channels. Not surprisingly, the number of Web-based vulnerabilities and attacks has also increased in-step with this trend. Leading this wave of Web-based vulnerabilities and attacks is cross-site scripting. This presentation demonstrates two types of cross-site scripting vulnerabilities (persistent and non-persistent) and sample mitigations that demonstrate how encoding reduces the risk from cross-site scripting attacks. The code included in this lab is for demonstration purposes. In production environments, please use the Microsoft Anti-Cross Site Scripting library that is available from [http://www.microsoft.com/sdl](http://go.microsoft.com/?linkid=9672761).

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

## Presentation Voiceover

A voiceover of the Cross-Site Scripting Vulnerabilities (Level 200) presentation transcript below, approximately 44 minutes in length, is also available to assist the presenter in becoming sufficiently acclimated with the subject matter addressed in the Cross-Site Scripting 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: Cross-Site Scripting Vulnerabilities](http://go.microsoft.com/?linkid=9672756)

### Slide 2 – Title Slide

The Cross-Site Scripting 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 Web-based vulnerabilities encountered today called cross-site scripting, as well as a discussion on how the Microsoft SDL can be applied to reduce exposure to 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, cross-site scripting vulnerabilities, and how exposure to attacks based on this particular type of vulnerability can be reduced 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 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 cross-site scripting vulnerabilities, and how the exposure to attacks based on these vulnerabilities can be reduced using the guidance, process and tools provided by the Microsoft SDL.

### Slide 5 – Cross-Site Scripting Overview

Cross-site scripting vulnerabilities, or “XSS” for short, occur whenever an application reads user data and embeds that data in Web responses without encoding or validating that data. That user data may have originated from an untrusted source, such as a malicious user or an untrusted database, and may contain malicious code or script. Client Web browsers, such as Internet Explorer, Firefox and Safari, will read these Web responses and execute any embedded code or script. As far as the Web browser can tell, the data contained in those Web responses originated entirely from the responding Web server, and all code and script is executed using the established trust level with that Web server.

Cross-site scripting gives malicious users the ability to perform nefarious actions against client Web browsers, such as stealing session data and modifying the appearance of loaded Web sites.

Some common vulnerabilities in Web-based applications that make cross-site scripting attacks possible include improper validation of input, not encoding Web responses, and trusting data read from shared resources. Encoding will be discussed later in this presentation, but briefly encoding is a way in which developers can transform potentially executable code or script into non-executable code or script.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on cross-site scripting, 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 – Cross-Site Scripting In The News

Here are some news stories regarding cross-site scripting.

* In October 2005, MySpace users were attacked with a cross-site scripting-based worm that infected over a million users within several hours. A story about this attack can be found at <http://www.securecomputing.net.au/News/72436,myspace-superworm-creator-sentenced-to-probation-community-service.aspx>.
* In 2007, cross-site scripting vulnerabilities were reported by the Open Web Application Security Project (OWASP) as the most common Web application vulnerability. As companies continue to increase reliance on the Web for business enablement, the frequency of cross-site scripting attacks in the coming years is expected to grow. The OWASP top 10 report for 2007 can be found at <http://www.owasp.org/index.php/Top_10_2007>.
* On June 26, 2008, Secure Computing magazine reported that a cross-site scripting vulnerability in Yahoo Mail could allow a malicious user to steal user’s Yahoo credentials. This report can be found at <http://www.securecomputing.net.au/News/115261,yahoo-mail-crosssite-scripting-flaw-targets-im-users.aspx>.

### Slide 7 – Types of Cross-Site Scripting

Two types of cross-site scripting vulnerabilities exist. Both result in malicious code or script being executed on the client-side browser; however, they differ in the method in which that malicious code or script is delivered to the client.

**Type 1: Non-Persistent (or Reflected)**

The first type of cross-site scripting is called type 1 or “non-persistent cross-site scripting”. You may often hear this type of cross-site scripting referred to as “reflected cross-site scripting”. With non-persistent cross-site scripting, malicious code or script is embedded in a Web request, and then partially or entirely echoed (or “reflected”) by the Web server without encoding or validation in the Web response. The malicious code or script is then executed in the client’s Web browser which could lead to several negative outcomes, such as the theft of session data and accessing sensitive data within cookies. In order for this type of cross-site scripting to be successful, a malicious user must coerce a user into clicking a link that triggers the non-persistent cross-site scripting attack. This is usually done through an email that encourages the user to click on a provided malicious link, or to visit a web site that is fraught with malicious links.

**Type 2: Persistent (or Stored)**

The second type of cross-site scripting is called type 2 or “persistent cross-site scripting”. This type of cross-site scripting is often also referred to as “stored cross-site scripting”. Persistent cross-site scripting is the more dangerous of the two types of cross-site scripting. With this variant, malicious code or script is stored in a persistent data store, such as a database, via a vulnerable Web-based application. Examples include, but are not limited to, discussion forums, guest books, and any other platform where one user can share information with many others. Any user visiting this vulnerable Web-based application is automatically sent the stored malicious code or script and is subsequently compromised. This variant of cross-site scripting has the potential to compromise many users via a single attack. This is quite different from type 1 cross-site scripting where one attack can result in a maximum of a single compromise.

In the following slides, each of these types of cross-site scripting attacks and the attack scenarios they exercise will be discussed in more detail.

Note: Some resources may reference a third type of cross-site scripting called “type 0” or “DOM-based cross-site scripting”. “DOM” stands for Document Object Model, and with this type of attack a malicious user is specifically targeting client-side script. The malicious code or script delivery methodology for this type of cross-site scripting is very similar to type 1 cross-site scripting, and therefore type 0 is included with type 1 in this discussion.

### Slide 8 – Type 1: Non-Persistent Cross-Site Scripting

Let’s first take a look at a type 1 cross-site scripting attack, or sometimes referred to as a “reflective cross-site scripting attack”.

In this first scenario, we have a malicious user, a user who will be attacked, and a cross-site scripting vulnerable Web-based application which the malicious user will use to attack the user.

(Mouse click)

Here, the malicious user crafts a malicious email that is sent to the user.

(Mouse click)

Contained in the email is a message that indicates that the user has won a prize and that in order to claim that prize the user needs to click on the provided link. Unknown to the user, the link contains some malicious code that will be reflected back from the Web-server and executed on the user’s Web browser.

(Mouse click)

The unsuspecting user clicks on the link and a Web request containing the malicious code is sent to the Web server.

The Web server takes the malicious code set by the malicious user, and embeds it (or reflects it) somewhere in the resulting Web response without encoding or validation. The user’s browser reads the response and executes the code which then compromises the user in some way.

(Mouse click)

After the attack is completed, the reflected malicious code does not stay resident on the Web server. This is why this type of cross-site scripting attack is called “non-persistent cross-site scripting”. During the attack, you may have noticed that the user had to first be coerced into clicking the suspect link. Due to this required user interaction for this attack to succeed, many development teams disregard this type of attack claiming that, “users would not do that.” The reality, however, is there is a demonstrated and ever-growing trend of unsuspecting users clicking on malicious links presented to them. This is reinforced by the fact that spam messages continue to be so prolific today. Links like these may raise the suspicion of more tech-savvy and security-aware individuals like developers and security professionals; however, for everyone else, links like these probably would not raise suspicion and more than likely will be clicked.

This is how type 1 non-persistent cross-site scripting attacks work. In the next slide, you will see how type 2 persistent cross-site scripting attacks work.

### Slide 9 – Type 2: Persistent Cross-Site Scripting

The final variant of cross-site scripting attacks is type 2 persistent cross-site scripting, or sometimes to referred to as “stored“ or “second-order cross-site scripting” attacks. This version of cross-site scripting is the most dangerous of all types of cross-site scripting attacks because, unlike type 1, type 2 cross-site scripting attacks do not require user interaction in order to trigger the delivery of malicious code.

(Mouse click)

In this scenario, we have a malicious user, a Web server that hosts a Web-based application that is susceptible to persistent cross-site scripting attacks, and a database that is used by the Web-based application to store information, such as user names, blog comments, and other user-provided data.

(Mouse click)

The malicious user injects malicious code or script into a vulnerable input field, such as a blog comment, and submits the page request.

(Mouse click)

The Web application then stores the blog comment into the back-end database.

(Mouse click)

Now, any user that visits this Web site, accesses the Web-based application, and attempts to view the blog comments will cause the Web-based application to retrieve the stored malicious code as part of the Web response to the user.

(Mouse click)

The user’s Web browser reads the response along with the malicious code and is compromised. Any other user who tries to access this Web-based application in a similar fashion is also automatically compromised.

Persistent cross-site scripting attacks are particularly dangerous because they have the potential with one attack to compromise many users. This is in contrast to the previous cross-site scripting variant (type 1) where one attack corresponds to one compromised user. Furthermore, this attack did not rely on users being coerced into clicking some malicious link.

A final note about both types of cross-site scripting before moving onto a demonstration: Since the injected malicious code or script appears to originate from the vulnerable Web-based application from the client Web browser’s perspective, that malicious code or script will have access to any session data set by that Web-based application, such as potentially sensitive data stored in cookies.

### Slide 10 – Cross-Site Scripting Demo: The Contoso Credit Union

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

Please note that several security controls that are automatically enabled on Microsoft platforms had to be disabled for this demonstration; however, the goal of the demonstration is to visualize the potential for damage when those security controls and best practices are not used or are disabled.

(Start SDL Demonstration Platform Contoso Credit Union cross-site scripting attack demonstration)

### Slide 11 – Common Cross-Site Scripting Myths

Several myths about cross-site scripting vulnerabilities exist. Here are two of the most common ones.

(Mouse click)

**Myth #1: Cross-site scripting applies only to Web-based applications built on Microsoft technologies.**

The first common myth is that cross-site scripting applies to only Web-based applications built on Microsoft technologies, such as ASP, ASP.NET, or that are hosted on Microsoft Internet Information Server (IIS). Cross-site scripting is a client-side attack that targets the Web browsers being used to access a Web-based application. Any Web browser that reads HTML, JavaScript, or any other Web-based language is susceptible to cross-site scripting.

(Mouse click)

**Myth #2: Cross-site scripting can be remedied by using transport security protocols, such as SSL and IPsec.**

The second common myth regarding cross-site scripting is that attacks of this nature can be remedied through the use of transport security protocols, such as SSL or with IPSec. This is a myth because cross-site scripting is an application-level attack that is not affected in any way by the underlying transport method.

### Slide 12-13 – Reducing the Exposure to Cross-Site Scripting Attacks

There are several measures you can take as a developer to reduce the exposure to cross-site scripting attacks conducted through your Web-based applications.

(Mouse click)

The first defensive measure which can be applied to address a majority of application security vulnerabilities is input validation. Validating input is a key design, develop and test tenet of the SDL and should be used to ensure that all untrusted inputs into Web-based applications conform to the expected input formats. Check for correctness with format, length, type, and range. Example sources of untrusted input include, but are not limited to, data from users, data from a database, or data from an un-trusted Web service.

More information on validating input in ASP.NET can be found at http://msdn.microsoft.com/en-us/library/ms972961.aspx.

(Mouse click)

The second defensive measure is to encode any Web response data that may contain user input or other untrusted input. Encoding will be discussed in more detail later in this presentation, but briefly encoding in the context of cross-site scripting attacks works by taking data that may contain executable code or script, and transforming or neutralizing it into non-executable forms.

More information about encoding Web response data to prevent cross-site scripting attacks can be found at <http://msdn.microsoft.com/en-us/library/ms998274.aspx>.

(Mouse click)

Web-based applications built using Microsoft ASP.NET can leverage built-in protection via the ValidateRequest option. This option, when set to “true,” instructs ASP.NET to inspect all inputs into a Web-based application for potentially dangerous inputs. If any potentially dangerous inputs are detected, then an HttpRequestValidationException is thrown and the attack is halted. This feature can be enabled on a per-page basis, or globally through web.config file settings. It is important to point out that this feature provides only limited protection, and should be used in conjunction with safe development practices, such as input validation and output encoding.

More information about the ValidateRequest option can be found at <http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.validaterequest.aspx>.

The remaining defensive measures are relevant to specific Web-based application and browser scenarios; however it is still important to be aware of these.

(Mouse click – Next Slide)

In ASP.NET Web-based application scenarios where access to cookie data needs to be protected from client-side attacks, the System.Web.HttpCookie.HttpOnly property may be used. When set to “true”, this option limits any access to cookie data via client-side scripts. Please note that this option will protect against any client-side access attempts to cookie data, but does not protect the cookie data in transport. For this scenario, use SSL or another transport security protocol in addition to the HttpOnly option.

More information about the HttpOnly property can be found at <http://msdn.microsoft.com/en-us/library/system.web.httpcookie.httponly.aspx>.

The next defensive measure that will be discussed in this presentation are the <frame> and <iframe> tag security attributes. For users using Internet Explorer 6 and higher, Web-based application developers can make use of the <frame> and <iframe> security attributes to limit embedded content within frames and iframes from executing potentially malicious scripts.

More information about the <frame> and <iframe> security attributes can be found at <http://msdn.microsoft.com/en-us/library/ms534622.aspx>.

The final defensive measure that can be used to help protect applications from cross-site scripting attacks is the Microsoft Anti-Cross Site Scripting Library (AntiXSS). This library provides additional encoding capabilities not provided by the standard encoding libraries included in the .NET Framework. More information about this library can be found at <http://www.codeplex.com/AntiXSS>.

A more in depth discussion of these defensive measures can be found at the Microsoft Patterns & Practices site at <http://msdn.microsoft.com/en-us/library/ms998274.aspx>.

### Slide 14 – How Encoding Works

In the previous slides, encoding was highlighted as one of the key defensive measures developers can use to reduce exposure to cross-site scripting attacks. The idea behind encoding works as follows.

(Mouse click)

To reduce the risk from cross-site scripting attacks, developers need to transform or neutralize user input that may contain potentially executable code or script into non-executable forms. That is, the Web browser needs to be told in some way that the following data is not executable code and should treated as data only. The way this transformation or neutralization is achieved is through encoding.

(Mouse click)

Developers building Web-based applications can use the following encoding libraries to help reduce the risk from cross-site scripting attacks.

(Mouse click)

The .NET Framework has built-in encoding libraries under the class System.Web.HttpUtility. The encoding methods in this class work by looking for specific characters that are common in cross-site scripting attacks and encode them into non-executable forms.

(Mouse click)

The Microsoft Anti-Cross Site Scripting Library takes a different approach by first defining a set of valid characters, and then encoding any characters not in that valid set. Both are effective in reducing exposure to a majority of cross-site scripting attacks; however, they differ in the method in which they reduce exposure.

More information about the HttpUtility encoding methods can be found at: <http://msdn.microsoft.com/en-us/library/system.web.httputility.aspx>.

More information about the Microsoft Anti-Cross Site Scripting Library can be found at: <http://msdn.microsoft.com/en-us/library/aa973813.aspx>. At the time of preparing this presentation, the most recent and recommended version of the Microsoft Anti-Cross Site Scripting Library was version 1.5.

### Slide 15 – Encoding Example

Let’s see an example of code where untrusted data is read from the user and then echoed back in a Web response without encoding. As you will see, encoding data bound for Web responses is easy using the built-in .NET and ASP.NET encoding libraries.

(Mouse click)

In the code you see here, a query string parameter called PhoneNumber is being read. That read value is used to set the Text property of a Label control called PhoneNumberLabel. If that value contains any executable malicious code or script, then the Web response will also contain that executable malicious code or script, and the recipient will be compromised.

(Mouse click)

As a developer, you can easily remedy this vulnerability by encoding the PhoneNumber value before using it to set the PhoneNumberLabel control text property. Here you see two examples of encoding. One with the standard encoding library included in the .NET Framework, and the second that is commented out with the newer Microsoft Anti-Cross Site Scripting Library.

When you are encoding untrusted data, be sure to encode the data just as it is about to be written to Web responses. Also be sure not to encode data more than once as this may interfere with the functionality of your Web-based application. Finally, be cautious of any data that you are decoding since this may return untrusted data back to an executable state.

Let’s see a demonstration of how encoding and other ASP.NET security controls could have been used to easily reduce the exposure to the cross-site scripting attacks demonstrated earlier.

### Slide 16 – Reducing Exposure to Cross-Site Scripting Attacks Demonstration

As seen in the previous slides, protecting Web-based applications and users against cross-site scripting attacks requires that developers utilize a series of defensive measures and best practices. Furthermore, relying on just a single defensive measure alone may not be sufficient. In this coming demonstration, we will see how the Contoso Credit Union code can be modified to use cross-site scripting attack defensive measures such as encoding and the ASP.NET’s ValidateRequest feature.

(Start SDL Demonstration Platform Contoso Credit Union cross-site scripting attack exposure reduction demonstration)

### Slide 17 – Cross-Site Scripting Scanning Tools

In addition to safer coding practices to reduce the exposure to cross-site scripting attacks, you should also be regularly reviewing your code for this type of vulnerability. Any section of Web-based application code that entirely or partially echoes untrusted data in Web responses is suspect for cross-site scripting. Microsoft has published a code scanning tool, called XSSDetect, which integrates with Visual Studio to assist developers and testers in this endeavor. It should be noted that this tool is designed to assist in code review and is not designed to replace/obviate the overall code review process. This tool is effective in identifying only certain coding patterns that can lead to cross-site scripting vulnerabilities and nothing else; this tool should be combined with expert manual code review and other security verification practices.

**Microsoft XSSDetect (Beta):** Microsoft XSSDetect is a static code analysis tool that helps identify Cross-Site Scripting security vulnerabilities found within Web-based applications. It is able to scan compiled managed assemblies (i.e., C#, Visual Basic .NET, and J#) and analyze dataflow paths from sources of user-controlled input to vulnerable outputs. Microsoft XSSDetect can be downloaded at <http://www.microsoft.com/downloads/details.aspx?FamilyID=19a9e348-bdb9-45b3-a1b7-44ccdcb7cfbe&displaylang=en>.

### Slide 18 – Conclusion

This concludes the discussion on cross-site scripting vulnerabilities. Cross-site scripting vulnerabilities are the most frequently encountered Web-based vulnerabilities today, and have been found on several major Web sites. These vulnerabilities manifest in Web-based applications whenever best practices, such as input validation, and Web output encoding are not implemented in code. To reduce exposure to these attacks, developers should implement a multi-layer defense strategy that includes coding best practices such as input validation, Web output encoding, and leveraging built-in platform protection. Microsoft has better enabled developers to do so through the guidance, process and tools of the Microsoft SDL.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation which focused on cross-site scripting, 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 – Reducing Exposure to Cross-Site Scripting Attacks using ASP.NET

The Microsoft Pattern & Practices Web site contains detailed information on reducing the exposure to cross-site scripting attacks using ASP.NET at <http://msdn.microsoft.com/en-us/library/ms998274.aspx>.