MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

SECURITY CODE REVIEW (LEVEL 200)

Guide

Version 1.0

The following documentation provides presenter’s notes for the Microsoft Security Development Lifecycle (SDL) Security Code Review (Level 200) presentation.

For the latest information, please see <http://www.microsoft.com/sdl>.

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 Field Content 2](#_Toc233625678)

[1.1 Introduction 2](#_Toc233625679)

[1.2 System Requirements 2](#_Toc233625680)

[1.3 Presentation Themes 2](#_Toc233625681)

[2.0 Microsoft SDL Security Code Review 2](#_Toc233625682)

[Overview 2](#_Toc233625683)

[Presentation Transcript 2](#_Toc233625684)

[Presentation Voiceover 2](#_Toc233625685)

[Presentation Demonstrations 3](#_Toc233625686)

[Slide 2 – Title Slide 3](#_Toc233625687)

[Slide 3 – Agenda 3](#_Toc233625688)

[Slide 4 – Microsoft Security Development Lifecycle (SDL) 3](#_Toc233625689)

[Slide 5 – Security Code Review Overview 4](#_Toc233625690)

[Slide 6 – Security Code Review Illustrated 5](#_Toc233625691)

[Slide 7 – The Importance of Security Code Review 5](#_Toc233625692)

[Slide 8 – Security Code Review Pros and Cons 5](#_Toc233625693)

[Slide 9 – Security Code Review Tips 6](#_Toc233625694)

[Slide 10 – Integrating Security Code Review into Software Development Lifecycles 7](#_Toc233625695)

[Slide 11 – Security Code Review Examples 8](#_Toc233625696)

[Slide 12 – Example: Buffer Overflow 8](#_Toc233625697)

[Slide 13 – Example: SQL Injection 9](#_Toc233625698)

[Slide 14 – Example: Cross-Site Scripting 9](#_Toc233625699)

[Slide 15 – Microsoft SDL Security Code Review Recommendations 9](#_Toc233625700)

[Slide 16 – Conclusion 10](#_Toc233625701)

[Slide 17 - Appendix 10](#_Toc233625702)

[Slide 18 – Microsoft Security Development Lifecycle (SDL) 11](#_Toc233625703)

[Slide 19 – Microsoft Writing Secure Code Book Series 11](#_Toc233625704)

[Slide 20 – Microsoft Developer Network (MSDN) Security Developer Center 11](#_Toc233625705)

[Slide 21 – Secure Development Blogs 11](#_Toc233625706)

[Slide 22 – Hunting Security Bugs 11](#_Toc233625707)

[Slide 23 – Additional SDL Training 11](#_Toc233625708)

# 1.0 Security Development Lifecycle Field 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:

* Security Code Review (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 Security Code Review

## Overview

Building safer and more trusted applications that can better withstand attack from today’s threat landscape requires a strong security strategy. A highly effective method that should be part of almost every security regimen is security code review.

This presentation provides an overview of the security code review process, security code review examples for common vulnerabilities and security code review recommendations provided by the Microsoft SDL.

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

## Presentation Voiceover

A voiceover of the Security Code Review (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 Security Code Review (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: Security Code Review](http://go.microsoft.com/?linkid=9672758)

### Slide 2 – Title Slide

The Security Code Review (Level 200) presentation introduces the role that the Microsoft Security Development Lifecycle (SDL) fulfills in trusted application development. It also provides an overview of the security code review process, which is the manual inspection of application source code for security vulnerabilities. Examples of how to review code for common security vulnerabilities and security code review recommendations from the Microsoft SDL are also presented.

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

Note: It is recommended that the presentations addressing the following topics be reviewed / addressed prior to this presentation:

* Buffer Overflows
* SQL Injection
* Cross-Site Scripting

### Slide 3 – Agenda

In this presentation, we will complete an overview of the Microsoft SDL, the security code review process, security code review examples addressing buffer overflow, SQL injection, and cross-site scripting concerns and security code review recommendations prescribed by 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 issues with security implications is not sufficient. Design vulnerabilities can have a substantial detrimental impact on security and are much more difficult to address during the verification phase. To that end, threat modeling is a critical SDL requirement and a Microsoft security innovation that is recognized by analysts as the next evolution in creating more secure software. Through threat modeling, architects and developers at Microsoft are able to approach security in a structured and methodical way from an attacker’s perspective. This allows Microsoft to identify and reduce the attack surface and mitigate the risk of potential security design issues.

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

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

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

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

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

As was previously indicated, this presentation focuses on security code review and how it can be used to complement other security assessment activities as prescribed by the Microsoft SDL. With respect to specific phases of the Microsoft SDL, this presentation focuses on the Implementation and Verification phases.

### Slide 5 – Security Code Review Overview

Building safer and more trusted applications that can better withstand attack from today’s threat landscape requires a strong security strategy. One highly effective method that should be part of almost every security regimen is the security code review process. Security code review involves the manual inspection of an application to identify security vulnerabilities at the code level.

Security code reviews are best conducted when an application is nearing a stable state and the possibility of future code changes are minimal. This is recommended so as to minimize the amount of wasted efforts that application development teams will have to expend reviewing code that will significantly change. It is in part for this reason why security code reviews are conducted towards the final stages of the Microsoft SDL Verification phase called the “Security Push”. More about this important phase will be discussed later in this presentation.

Oftentimes security code reviews are performed solely by security analysts. However, given the size and complexity of modern applications, security testing should ideally be performed by developers, testers *and* security analysts. Application team developers and testers provide familiarity with the application’s design and implementation that can aid in identifying vulnerabilities that may be contrary to the overall intended functionality. Security analysts can perform security code review efforts that are free from assumptions and previously established biases in order to help identify vulnerabilities that are outside the overall intended functionality of the application.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation focusing on security code review, 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 – Security Code Review Illustrated

If you are new to the concept of security code review, the easiest way to conceptualize this security assessment process is to imagine the role of a publication editor.

(Mouse click)

When an author submits written text, such as a book, news article or whitepaper for publication, an editor will review that text prior to release. This is to ensure that the final publication is of the highest possible quality. As part of the editorial process, each line of the text is reviewed for errors, such as grammatical errors, inappropriate language usage and other aspects that might affect the overall quality of the text. Feedback from the editors is provided back to the authors and they amend the text as necessary.

(Mouse click)

In the same fashion that an editor will review written text for errors, developers, testers and security analysts review application code for vulnerabilities that could negatively affect the quality of the application ultimately being delivered to customers. Vulnerabilities, such as buffer overflows, SQL injection and cross-site scripting are flagged by review teams and addressed by developers prior to releasing the application.

### Slide 7 – The Importance of Security Code Review

With the availability of automated code analysis tools, such as Microsoft FxCop and Microsoft PREFast, which can scale and analyze code much faster than human reviewers, a question that you may have is, “Are manual security code reviews really necessary?”

Any assessment method, including the use of code analysis tools, has specific strengths and weaknesses. In the case of code analysis tools, certain vulnerabilities may be easily detected while other types of vulnerabilities are difficult to detect. There may be a gap between risks that are detectable by code analysis tools and actual risk present. No one assessment method by itself is sufficient to comprehensively assess the security posture of an application. This is why the Microsoft SDL implements a variety of different assessment methods at various stages throughout the software development lifecycle to help development teams deliver applications that are safer and more trustworthy.

To illustrate this point let’s look at a demonstration of a code vulnerability that is difficult for code analysis tools to detect, but easily detectable through conducting a security code review.

(Start demonstration)

### Slide 8 – Security Code Review Pros and Cons

Security code reviews, like any other risk assessment method, has several pros and cons.

(Mouse click)

One key advantage of security code reviews is that a deeper analysis is possible than that which is possible with other security assessment methods. Since the code is entirely visible to the reviewer, the need to make certain assumptions that other methods may apply is reduced, and thus the number of false positives and false negatives are also reduced. Reviewers can also draw upon their personal experiences and expertise in certain technologies that may be difficult for automated methods to emulate. Essentially, the security code review process will be able to identify certain vulnerabilities that other assessment methods cannot.

(Mouse click)

Another advantage of security code reviews is that assessments can be performed in a more context-aware fashion than that which is possible with machine-based methods. Human reviewers have the distinct advantage in that they are able to interact with the architects and developers of an application. Through this interaction they can gain a better understanding of the application, such as the intent of the application, valid user scenarios and dependencies, and make more meaningful reports and recommendations. Having this understanding allows reviewers to better determine critical components of the application and focus efforts on those areas rather than on less critical components.

(Mouse click)

In addition to the advantages of employing a security code review process, there are certain disadvantages of which you should be aware. One key limitation of the security code review process is that it is a manually-intensive process. Human error and fatigue may become factors when reviewing application code, especially when performing a security code reviews for large applications. Furthermore, with modern day applications easily exceeding hundreds of thousands, and sometimes millions of lines of code, completing a security code review may require a large amount of time and effort, therefore becoming a very expensive proposition.

(Mouse click)

The final notable limitation regarding the security code review process is that the quality of the review is highly dependent upon the skill level and expertise of the reviewer. With code analysis tools, for example, findings are repeatable and can be arrived at objectively. However, when conducting a security code review, findings are not necessarily repeatable and are not always identified in an objective manner. For example, a security code reviewer may be more skilled at finding vulnerabilities in one programming language versus others, or they may be more proficient with a particular type of vulnerability than others. Factors like these will affect the overall quality and sufficiency of the security code review process.

(Mouse click)

As a final note regarding the advantages and disadvantages of security code reviews, security code reviews like any other method for identifying risk in applications is not a silver bullet. Security code reviews can help you find many security vulnerabilities, but not all possible vulnerabilities. It should never be used as a replacement for other security assessment methods, but rather in conjunction with other methods.

### Slide 9 – Security Code Review Tips

Several of the disadvantages of the security code review process discussed in the previous slide can be addressed by the following security code review tips.

(Mouse click)

The first tip is to use a multi-pass approach. Many code reviewers make the mistake of trying to conduct security code reviews in a single pass. Certainly security code reviews can be performed in this manner; however, with larger applications doing so may prove very difficult. Fatigue tends to set in after processing large segments of code, which often increases the chances of errors being introduced into the security code review process.

As the name suggests, with a multi-pass approach, application code implementations are reviewed multiple times. Each pass could have a specific objective, such as looking for a particular type of vulnerability. Since reviewers are looking for specific vulnerabilities, such as buffer overflows, SQL injection and cross-site scripting rather than all vulnerabilities, each pass can be conducted more quickly and the likelihood of reviewers suffering from fatigue can be reduced. Furthermore, each pass could also be performed by different reviewers to further reduce the likelihood of human error.

(Mouse click)

The second tip is to focus security code review efforts on high-priority code, and then focus security code review efforts on less critical code as time permits. Since security code reviews can become very expensive, prioritizing security code review efforts is an important way to help ensure that critical components have been thoroughly reviewed prior to customer release while staying within available budgets. Further discussion regarding prioritizing application code will be discussed later in this presentation.

(Mouse click)

One common criticism about the security code review process is that it is often performed in an ad-hoc manner. With the ad-hoc approach, reviewers take an application and begin randomly looking for vulnerabilities without any clear objectives. Certainly security code reviews can be performed in this manner; however, in the absence of documented goals, it is difficult for application development teams to understand the level of coverage that is provided by security code reviews. Many questions begin to surface, such as, “Which vulnerabilities have been considered when conducting the security code review?” “To what extent were the vulnerabilities tested?” “Specifically which vulnerabilities were not investigated when conducting the security code review?” All these questions are difficult to answer when security code reviews are performed in an ad-hoc fashion. By first establishing clear security code review objectives and then manually reviewing code against those objectives, application developers can gain a more accurate understanding of the security posture of their applications. Application teams can then bring in other security assessment methods, such as code analysis tools, to address other objectives not evaluated by security code review efforts.

Example security code review objectives may be to ensure the following:

* That application code does not ignore return values;
* That all pointers are checked for validity; and
* That all un-trusted data is validated prior to use.

### Slide 10 – Integrating Security Code Review into Software Development Lifecycles

As previously mentioned, security code reviews are not meant to replace other security assessment activities and should be used as a complementary activity. Code reviews by themselves can be an effective means to identify security vulnerabilities in code; however, the benefits of security code reviews are amplified when they are done in the context of other security assessment activities. In fact, when security code reviews are performed in conjunction with other assessment activities, the complexity of performing security reviews can also be greatly decreased, therefore reducing the total expense associated with conducting the security code review process.

(Mouse click)

For instance, using automated code analysis tools, as recommended by the Microsoft SDL, could be performed prior to expending manual review efforts. Using automated code analysis tools introduces benefits by providing a higher degree of code review coverage. Code analysis tools can also help identify more common and frequently encountered vulnerabilities called “low hanging fruit.” Reviewers can then allocate their efforts to address more critical components of the application and to identify more complex and less apparent vulnerabilities.

Automated code analysis tools may also provide early indication as to the types of vulnerabilities that are most prevalent in the application code. Security code reviewers can use data from code analysis tools to better define security code review objectives and more effectively focus their efforts during their review process.

(Mouse click)

Another example might be to create threat models prior to using code analysis tools and performing security code reviews. Threat models are used to decompose an application and can help developers, testers and security analysts to better understand the internals of an application. Threat models also expose critical areas of an application that are most prone to attack. Security code reviewers could take that knowledge and focus their efforts on high-risk components of the application and expend less effort on components with minimal risk.

(Mouse click)

Security training materials could be updated to address the types of vulnerabilities encountered through employing a security code review process. This would help better ensure that developers do not introduce the same vulnerabilities into future code.

(Mouse click)

Finally, the lessons learned from performing security code reviews, such as the types of vulnerabilities that are most prevalent for a particular application, could be used to update internal processes. This would help to ensure that occurrences of those prevalent vulnerabilities are reduced in future releases of that particular application.

### Slide 11 – Security Code Review Examples

Let’s now look at some examples of how to review application code for some of today’s most common security vulnerabilities, including buffer overflows, SQL injection and cross-site scripting. The examples presented in the following slides represent common ways by which these types of vulnerabilities manifest in code and is not meant to be an all-encompassing guide.

Presentation materials that discuss each of these vulnerabilities in more detail are available and should be reviewed accordingly if you have not already reviewed their contents.

### Slide 12 – Example: Buffer Overflow

Buffer overflows can manifest in a myriad of different ways. Unsafe function calls and unsafe custom code that writes into buffers are some of the common ways that buffer overflows can be present in code. Does this mean that security code reviewers need to know all the possible ways that buffer overflows can be created? Fortunately, the answer is a resounding, “No.”

All buffer overflows, regardless of the technology or platform being used, manifest whenever an application attempts to write data into a fixed-length buffer beyond the bounds of that buffer. By looking for this pattern in code, security code reviewers can more easily find buffer overflows in any platform or technology.

(Mouse click)

Here is some sample C code that implements a function called *CopyBuffer*. This function reads a *char* ‘\*’ and allocates a fixed-length 32 byte buffer called *Buffer*. CopyBuffer then copies the *src* argument into Buffer using a call to *strncpy* and restricts the size of that copy to 32 bytes. Finally, CopyBuffer null-terminates Buffer by inserting a null-terminating character at the last index of Buffer. Can you see what is wrong with this code?

If the problem is not immediately apparent to you, try applying the pattern above to this code.

(Mouse click)

First, do we have a fixed-length buffer that will be written into anywhere in this code? Yes, the destination buffer called Buffer is one that fits this criterion.

(Mouse click)

Next, will the code ever try to write beyond the boundary of Buffer? The answer again is yes, but not with the call to strncpy. The overflow occurs when the function tries to null-terminate Buffer. Take a close look at which element index in Buffer gets null-terminated. According to the code, the index at *sizeof(Buffer)* gets null-terminated. The call to sizeof(Buffer) returns 32; however, buffers in the C/C++ language start at zero and not at one. This means that the code is writing a null-termination character exactly one byte beyond the bounds of the fixed-length buffer called Buffer! This is what is known as an off-by-one buffer overflow and it is exploitable!

### Slide 13 – Example: SQL Injection

SQL injection is another common vulnerability. SQL injection vulnerabilities are created in code whenever an application tries to execute a SQL query that has been dynamically using un-trusted data. Similar to the buffer overflow discussion, let’s apply this pattern to some sample code to uncover a potential SQL injection vulnerability.

(Mouse click)

Here is some sample C# code that defines a method named *RemoveEntry*. RemoveEntry accepts two arguments: a string named *EntryName* and a *SqlConnection* object called *myConnection*. RemoveEntry creates a query to remove a given entry by name and then executes that query using some database server bound to the given SQL connection object. Can you see the potential SQL injection vulnerability in this code?

If the problem is not immediately apparent to you, try applying the pattern above to this code.

(Mouse click)

First, is the method building a dynamically build SQL query using un-trusted data? Yes, it is. The variable *Query* is dynamically created using the input from the EntryName variable. This variable is provided into this function. Therefore, the contents of this variable cannot necessarily be trusted.

(Mouse click)

The second part of the SQL injection pattern is the execution of a dynamically built SQL query. Does this code do that? Yes it does! Therefore, since the method dynamically creates a SQL query using potentially un-trusted data and then tries to execute that query, you know that a SQL injection vulnerability exists in this code!

### Slide 14 – Example: Cross-Site Scripting

Another common security vulnerability frequently encountered in Web-based applications is cross-site scripting. In fact, cross-site scripting vulnerabilities are the most common vulnerabilities today. The common pattern among all cross-site scripting vulnerabilities is applications attempting to embed un-trusted data in Web responses without encoding the data.

(Mouse click)

Here is sample code from an ASP.NET application. The *Page\_Load* method shown here is the method that is called when the ASP.NET application is first loaded. The method reads the value associated with the query string parameter called *namekey*. It then checks to see if the namekey value is null or empty, and if it is not, it outputs a greeting using the value of namekey. Can you see the vulnerability in this method?

If the problem is not immediately apparent to you, try applying the pattern above to this code.

(Mouse click)

A cross-site scripting vulnerability occurs at the code highlighted. The value from namekey is un-trusted and the method Page\_Load embeds that data in a Web response without encoding it first.

### Slide 15 – Microsoft SDL Security Code Review Recommendations

Security code reviews are done as part of the Security Push phase of the Microsoft SDL. The Security Push is a team-wide effort that focuses on updating security documents, performing code reviews and conducting other risk identification activities. Security pushes begin after a product enters the Microsoft SDL Verification phase and when the product feature set and code implementation is nearing completion. Security code reviews are not restricted solely to this phase. It is also recommended that security code reviews be performed during development as code nears completion. Trying to condense a large number of code reviews in a short period of time, such as the Security Push phase, can negatively affect the quality and integrity of the overall security code review effort.

(Mouse click)

Security code reviews should be done on all Priority 1 code. Priority 1 code is code that is most prone to security vulnerabilities. Guidelines on how to classify code by priority is outlined in the [Microsoft SDL process guidance paper](http://go.microsoft.com/?linkid=9672762) and the [Microsoft SDL book](http://go.microsoft.com/?linkid=9672766). Any component of code that has experienced a large number of security vulnerabilities in the past is also considered Priority 1 code. The definition of “large” can be highly subjective and will differ from project to project. However, it is still important to review these components, as it is more likely that additional security vulnerabilities will be found in code previously having a large number of security vulnerabilities vs. components of code that historically have not had as many vulnerabilities.

(Mouse click)

If time permits, Priority 2 code should also be reviewed. Priority 2 code is optionally installed code that runs with user privileges. The Microsoft SDL also defines Priority 2 code as any code that does not meet Priority 1 criteria.

### Slide 16 – Conclusion

This concludes the discussion on security code review. Security code review is a manual process whereby developers, testers and security analysts review the source code of an application and make note of any coding patterns that could negatively impact the security posture of that application. At first glance, the security code review process may appear to be an unnecessary manual duplication of other automated assessment methods, such as code analysis tools. On the contrary, security code reviews are unique in that they take contextual information about an application into account during the review. This helps to enrich the overall review process and has the effect of producing deeper findings and findings that are more relevant to the actual intent of the application.

Security code reviews do not negate the need for other security assessment activities. In fact, the benefits from security code reviews are further amplified when security code reviews are performed in conjunction with and complementary to other security assessment activities. Prioritization information from threat models and results from code analysis tools can help to better refine security code review objectives and improve results. Information about the vulnerabilities identified during security code reviews can be used to update training materials and internal processes to help ensure the occurrence of those vulnerabilities are reduced over time.

While Microsoft’s ability to continually deliver safer and more trusted applications to its customers through the Microsoft SDL can in part be attributed to extensive security code review efforts, security code reviews, like any other assessment activity, have certain disadvantages. Being manually intensive and highly dependent on the individual skill-level of reviewers are some of the aspects that have prevented security code reviews from being broadly adopted. However, disadvantages such as these can be overcome by following best-practices, such as using a multi-pass approach, focusing on high-priority code first, and establishing clear objectives before initiating the security code review process. When these best practices are followed, security code reviews can become a very powerful asset in an application development team’s security strategy.

Lastly, the insights gleaned by Microsoft, which are incorporated in its SDL, and more specifically, in this presentation which focused on security code review, 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 17 - Appendix

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

### Slide 18 – 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 19 – 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 20 – 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 21 – 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 22 – 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 23 – 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.