MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

BUFFER OVERFLOWS (LEVEL 300)

Version 1.0

The following questions accompany the materials for the Microsoft SDL - Developer Starter Kit Buffer Overflows (Level 300) presentation.

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

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

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

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

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

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

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, SQL Server, 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 MICROSOFT SDL - DEVELOPER STARTER KIT CONTENT COMPREHENSION QUESTIONS 2](#_Toc233186625)

[1.1 INTRODUCTION 2](#_Toc233186626)

[2.0 BUFFER OVERFLOWS (LEVEL 300) QUESTIONS 2](#_Toc233186627)

# 1.0 MICROSOFT SDL - DEVELOPER STARTER KIT CONTENT COMPREHENSION QUESTIONS

## 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 has developed 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 questions that may be used to ascertain comprehension of the subject matter addressed within the Microsoft SDL Training Module: Buffer Overflows (Level 300) presentation. These questions have been designed to enable the presenter to ascertain the extent at which the participating personnel with application development responsibilities have comprehended the subject matter addressed in the Buffer Overflows (Level 300) training module, as well as enabling the presenter to assess participants’ ability to apply the subject matter addressed to practical secure and trustworthy application development scenarios.

# 2.0 BUFFER OVERFLOWS (LEVEL 300) QUESTIONS

**Question #1:** Why does the following code contain a buffer overflow vulnerability?

void SampleFunction(char \* str)

{

char \* Buffer = (char \*)malloc(32);

int index = 1;

while (str[index] != ‘\0’)

{

Buffer[index] = str[index];

index++;

}

Buffer[index]=’\0’;

}

1. The above code does not contain a buffer overflow vulnerability.
2. If the str parameter is greater than 32 bytes in length then the while loop can create an exploitable buffer overflow vulnerability.
3. Arrays in C are zero-based. Therefore, starting the index variable at 1 will result in a buffer overflow of 1 byte.
4. The variable Buffer is allocated on an application’s heap; however, the str parameter may have been allocated on the stack elsewhere, and the type mismatch may create a buffer overflow vulnerability.

**Answer:** The correct answer is “**B**”. The code makes the assumption that the length of the incoming parameter str is less than or equal to 32 bytes. More precisely, the occurrence of the first null-character (if any) within the incoming parameter str occurs within 32 bytes. If any of these assumptions fail, the while loop will create the same exploitable condition as would a dangerous call to strcpy.

**Question #2:** Buffer overflows that are exploited on application heaps work by overwriting which memory regions?

1. Buffer overflows are not exploitable on application heaps.
2. The forward and backward pointers of heap chunks adjacent to the overflowed buffer.
3. The data regions of heap chunks that contain the overflowed buffer.
4. The forward and backward pointers of the heap chunk that contains the overflowed buffer.

**Answer:** The correct answer is “**B**”. Buffer overflow attacks that exploit buffers allocated on heaps are done so by corrupting the forward and backward pointers of the heap chunks that are adjacent to the overflowed buffer.

**Question #3:** Buffer overflows affect applications written for which of the following operating systems?

1. Microsoft.
2. Linux.
3. Mac OS X.
4. All of the above.

**Answer:** The correct answer is “**D**”. Buffer overflows occur whenever data is written into a fixed-length buffer and the size of that data exceeds the capacity of the receiving buffer. This pattern often occurs, but is not limited to, in applications written in native languages, such as C and C++. Since these languages are implemented on all of the operating systems listed above, applications written for each of these operating systems are susceptible to buffer overflow attacks.

**Question #4:** Which of the following statements is true about the code shown below?

void SampleFunction(char \* str)

{

/\* Allocate 32 bytes heap space \*/

char \* Buffer = (char \*)malloc(32);

/\* Copy str into Buffer \*/

strcpy(Buffer,str);

}

1. The parameter str is guaranteed to be less than 32 bytes in length.
2. Strcpy will ensure that a maximum of 32 bytes is copied from str into the allocated buffer.
3. A buffer overflow vulnerability exists in the code shown below because the parameter str may contain more than 32 bytes of data and it is being written into a fixed-length buffer.
4. The code above does not contain a buffer overflow vulnerability because the buffer allocated exists on the heap and not the stack where exploitation is possible.

**Answer:** The correct answer is “**C**”. A buffer overflow vulnerability does indeed exist in the code above because str may contain more than 32 bytes and is written into a fixed-length buffer. Answer “A” is incorrect because no code is implemented to ensure that str is less than 32 bytes in length. Answer “B” is also erroneous because the C runtime function strcpy will terminate writing when it encounters the first null-termination character and provides no guarantee regarding the maximum number of bytes written. Finally, answer “D” is also incorrect because buffer overflows can occur and be exploited on both application stacks and heaps.

**Question #5:** Which of the following mitigation approaches, if implemented, would effectively reduce the risk from the buffer overflow vulnerability addressed in Question #4 above?

1. Running the application code in a low-privilege, non-administrator security context.
2. Allocating the overflowed buffer on the stack instead of the heap.
3. Compiling and running the application code on another operating system besides Microsoft Windows.
4. Replacing the call to strcpy with a call to strcat.

**Answer:** The correct answer is “**A**”. Running the application in a limited security context, such as a non-administrator account, reduces the potential damage from a buffer overflow attack. Any malicious code executed will be restricted within the rights granted to the limited security context. This is an example of implementing the Least Privilege design principle. Please note that running the application code in a low-privilege non-administrative context does not resolve the actual vulnerability. Developers should implement other controls to further reduce the risk from the affected code, such as validating the size of the input to ensure that the destination buffer has sufficient space for the source buffer. Additionally, developers could replace the call to the strcpy function with an equivalent function from safer libraries such as the Safe CRT (strcpy\_s) and StrSafe (StringCchCopy and StringCchCopyEx) libraries. Answer “B” does not reduce the risk from the vulnerable code because exploitation of a stack-based buffer overflow is possible. In fact, re-engineering the code to use a stack-based buffer overflow would increase the probability of a successful attack since stack-based buffer overflows are much easier to exploit than heap-based buffer overflows. Answer “C” is incorrect because buffer overflows can still occur on non-Microsoft operating systems. Finally, answer “D” is incorrect because concatenating data into an empty buffer has the same effect as copying data into a buffer; the buffer overflow vulnerability would still exist.

**Question #6:** True or false: A buffer that is allocated on an application’s memory heap is immune from buffer overflow attacks

1. True.
2. False.

**Answer:** The correct answer is “**B**” (False). While the exploitation technique for exploiting buffers allocated on the heap is more complicated than the technique for exploiting buffers allocated on the stack, heap-based buffer overflow vulnerabilities can still be present and exploited accordingly.