MS_logo_KMICROSOFT SDL - DEVELOPER STARTER KIT:

FUZZ TESTING (LEVEL 200)

Version 1.0

The following questions accompany the materials for the Microsoft SDL - Developer Starter Kit Fuzz Testing (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 Microsoft SDL - Developer Starter Kit Content Comprehension Questions 2](#_Toc233611223)

[1.1 Introduction 2](#_Toc233611224)

[2.0 Fuzz Testing (Level 200) Questions 2](#_Toc233611225)

# 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: Fuzz Testing (Level 200) 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 Fuzz Testing (Level 200) 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 Fuzz Testing (Level 200) Questions

**Question #1:** What types of vulnerabilities will not be typically identified through fuzz testing activities?

1. Stack-based buffer overflows.
2. Format string.
3. Unicode-based buffer overflows.
4. Vulnerabilities that do not result in an application crash or exception, such as information disclosure and cryptography-based vulnerabilities.

**Answer:** The correct answer is “**D**”. Fuzz testing works by sending malformed data to application data entry points and developers analyze any unexpected or unhandled crashes, exceptions and errors. Information disclosure vulnerabilities and cryptography-based vulnerabilities do not typically result in application crashes, and therefore are often not uncovered through fuzz testing activities. All other vulnerabilities listed above typically result in crashes and can be detected through fuzz testing activities.

**Question #2:** A network tool that sends random data to the open network ports of an application is an example of what type of fuzzer?

1. Smart fuzzer.
2. Dumb fuzzer.
3. Quick fuzzer.
4. Advanced fuzzer.

**Answer:** The correct answer is “**B**”. When a fuzzing tool sends random data to the data entry points of an application without regard to the expected format or type of data, it is a dumb fuzzer.

**Question #3:** With respect to network traffic fuzzing, what is the primary difference between these two techniques?

1. Creating malformed network packets, and
2. Creating malformed network packets on the fly?
3. Technique 1 sends malformed packets directly to the target application, whereas technique 2 intercepts valid packets, malforms them and then sends those malformed packets to the target application.
4. There is no difference; both refer to the same technique.
5. Technique 1 uses TCP, while technique 2 uses UDP as the transport protocol.
6. Technique 1 uses message-based fuzzing, while technique 2 uses stream-based fuzzing.

**Answer:** The correct answer is “**A**”. With technique 1, malformed packets are created and send directly to the target application, whereas with technique 2 valid packets are intercepted in a man-in-the-middle fashion, malformed and then sent to the target application.

**Question #4:** A fuzzing tool that captures valid packets that are sent to an application, malforms those captured packets and then resends them at a later time is an example of which network fuzzing technique?

1. Creating malformed network packets.
2. Record-fuzz-replay.
3. Creating malformed network packets on the fly.

**Answer:** The correct answer is “**B**”. Recording valid network traffic, malforming it and then resending the corrupted network traffic at a later time is characteristic of the record-fuzz-reply network fuzzing technique. This is in contrast to the technique listed in answer “A” where malformed network packets are sent directly to a target application, and in contrast to the technique listed in answer “B” where valid traffic is intercepted, malformed and then sent to the target application in real-time.

**Question #5:** An application development team has come to you for advice regarding fuzz testing. They are unsure as to which application build types they should be fuzz testing (release or debug). What application build type should they be fuzz testing and why?

1. Debug builds, because these builds can be run under a debugger.
2. Debug builds, because crashes in these types of builds are easier to detect.
3. Release builds, because the goal of fuzz testing is to identify vulnerabilities in applications while in operation, and therefore fuzz testing should be done on the same binaries that are delivered to end customers.
4. Release builds, because these builds will be optimized and will respond quicker to malformed data input attempts.

**Answer:** The correct answer is “**C**”. Fuzz testing aims to simulate an application under attack while in operation, and therefore the same binaries (release) that are delivered to end customers should also be fuzz tested.

**Question #6:** Which of the following methods and functions should be fuzz tested?

public byte[] CreateByteArray()

{

return(new byte[32]);

}

public void ReadDataFromNetwork()

{

TcpClient client = new TcpClient(“http://serveraddress”, 8080);

NetworkStream stream = client.GetStream();

byte[] data = new byte[256];

Int32 bytes = stream.Read(data,0,data.Length);

String ResponseData = System.Text.Encoding.ASCII.GetStrign(data,0,bytes);

// Perform some action with data

stream.Close();

client.Close();

}

public static byte[] ReadFile(String FilePath)

{

FileStream fs = new FileStream(FilePath, FileMode.Open, FileAccess.Read,

FileShare.None);

try

{

int length = (int)fs.Length;

byte[] data = new byte[length];

fs.Read(data,0,data.Length);

// Perform some action with the data read

return(data);

}

catch (Exception)

{

return(null);

}

finally

{

fs.Close();

}

}

1. None of the methods shown above require fuzz testing.
2. Only CreateByteArray requires fuzz testing.
3. CreateByteArray and ReadFile require fuzz testing.
4. All three methods require fuzz testing.
5. Only ReadFile requires fuzz testing.
6. ReadDataFromNetwork and ReadFile require fuzz testing, but not CreateByteArray.

**Answer:** The correct answer is “**F**”. Fuzz testing simulates the inputting of malformed data into application entry points. The method CreateByteArray neither takes any inputs, nor does it read from any application entry points. CreateByteArray therefore does not require fuzz testing. The method ReadDataFromNetwork does not read any input parameters; however, it does read data from a TCP connection and should be fuzz tested. The method ReadFile should also be fuzz tested because it reads data from an application entry point in the form of a given filename.