Microsoft

By Eugene Bobukh, Senior Security Program Manager, Security Assurance, Microsoft Security Engineering Center (MSEC), with contributions from the Silverlight team

November 4, 2009

For the latest information on the SDL, 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, ActiveX, Internet Explorer, Silverlight, SQL Server, 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.

**Introduction**

Security is indispensable for today’s software industry. According to U.S. government agencies, cybercrime is now more profitable than the wholesale illegal U.S. drug trade. As software vendors like Microsoft, Oracle, Red Hat, and other companies have worked hard to reduce the attack surface of their platforms, hackers have begun to focus on applications. Attacks are moving up the stack. At the same time, attacks are becoming much more targeted and stealthy.

The [*Microsoft Security Intelligence Report Volume 5 (January through June 2008)*](http://go.microsoft.com/fwlink/?LinkId=131912) shows that between the releases of Windows® XP and Windows Vista®, the percentage of Microsoft browser–based, exploited vulnerabilities among the top 10 has dropped from 50 percent to 0 percent. At the same time, the number of new vulnerabilities identified across the industry each six months has nearly quadrupled.

According to the [Microsoft Malware Protection Center](http://www.microsoft.com/security/portal/) assessment of the growing sophistication in the attacks they see, it is likely that hackers will pay even more attention to rich Internet applications (RIAs). Therefore, we think that over the next year and beyond, designers and developers of RIAs will select platforms that have the best security track record. They will choose RIAs that have the fewest identified vulnerabilities and that are architected from the ground up to provide enterprises and consumers with the best possible protection against malware and other security threats.

This paper reviews the Microsoft Security Development Lifecycle (SDL) requirements and the work that the Microsoft Silverlight™ team did to both meet and exceed those requirements for every phase of the Silverlight 1.0 release. These results are based on a review of the product’s security performance in the 12 months immediately following the release (September 4, 2007). Our experience has shown that a year is sufficient time for the security research community to gain familiarity with a new product or technology and to begin to identify and report the product’s residual vulnerabilities. In our experience with other products, such as Windows Server® 2003, Windows Vista, and Microsoft SQL Server® 2005, the first year’s experience has proven to be a good predictor of the product’s security over the long term.

Silverlight is a cross-browser, cross-platform browser plug-in for delivering rich user experiences, such as videos or vector graphics, on the Web. Since its release, Silverlight has been downloaded and installed on hundreds of millions of computers. Yet in more than one year following the ship date, no security vulnerabilities have been publicly reported in Silverlight 1.0.

We believe this is the direct result of the security work that was completed before Silverlight was released.

Note that in October 2008, a newer version of Silverlight—Silverlight 2—was released. That version reflects the experience and lessons learned from Silverlight 1.0, in addition to updates to the Microsoft Security Development Lifecycle and tools. Silverlight 2 has only been in the market for a short time, so we can't assess the success of its security yet. Therefore, only Silverlight 1.0 is considered in this document.

**OVERVIEW OF SILVERLIGHT 1.0**

Silverlight 1.0 is supported on Windows Internet Explorer®, Firefox, and Safari browsers on Windows or on Mac OS X. For Internet Explorer, Silverlight is implemented as a Microsoft ActiveX® plug-in; for other browsers, it is implemented as a plug-in within the corresponding models (for example, the npPlugin for Firefox).

Silverlight 1.0 offers the following functionalities:

* Vector and bitmap graphics rendering
* Video and audio playback
* UI creation and rendering based on XAML description
* User interactivity
* Some very limited scripting (through callbacks and exposure to Document Object Model [DOM])

Silverlight 1.0 offers the following design-level security boundaries:

* **Protection of the operating system resources.** Unlike common ActiveX controls, no Silverlight application gets general access to modify resources like files or registry keys.
* **Domain isolation.** Like most browsers, Silverlight maintains a cross-domain security boundary in its own actions.
* **Untrusted content handling.** Silverlight can withstand and parse untrusted content, such as media streams, tampered with while on the wire.

Silverlight also has an auto-update mechanism to provide fixes to customers smoothly and easily.

**SECURITY CONSIDERATIONS IN THE SILVERLIGHT 1.0 RELEASE**

Silverlight 1.0 faced several potential security challenges, some of them unique. These challenges were:

* **Exposure to untrusted data.** Silverlight is designed to play media received over the Internet.
* **Exposure through the browser.** This exposure opens many new Web-specific threats, such as cross-site scripting, domain isolation violations, and more.
* **Cross-platform and cross-browser capabilities.** These capabilities create a challenge because available tools and mitigations vary between the platforms.
* **Large number of customers.** Silverlight may have as many as several hundred million customers.
* **Auto-update functionality.** Getting the auto-update functionality right in terms of security is known to be a complex task. Although Windows Update and Microsoft Update have been successful in improving security, the Silverlight team decided not to depend on these technologies because Silverlight is a cross-platform product and because some customers choose not to use them.
* **Uniqueness of the product.** Silverlight was a completely new type of product for Microsoft. The company did not have significant history with creating such rich Web plug-ins and could not, therefore, rely upon past experience in this area.

**SPECIFICS OF APPROACH—WHY SECURITY IS DIFFERENT THIS TIME**

Because Silverlight was a new type of product for Microsoft and it was going to be highly exposed through the Internet, and due to the industry’s long record of security vulnerabilities in other products with similar functionality, the Silverlight team agreed that a proactive approach to security had to be a focus throughout the development process.

As a result of this understanding, many work items were identified early and carried through the release. Two guidance sources were used:

* Microsoft SDL. The Microsoft Security Development Lifecycle is the standard set of security best practices, processes, and tools applied to all major Microsoft products.
* A series of brainstorming sessions and workshops to identify areas over and above the SDL that should be explored because of the unique challenges that Silverlight presented.

**EXAMPLES OF SPECIFIC SDL ACTIVITIES**

Begun in 2002, the Microsoft SDL was designed to eliminate as many software vulnerabilities as practical and to help reduce the severity of vulnerabilities that are not eliminated. The SDL is a holistic and comprehensive approach that leverages education, process, and accountability to consistently create more secure software. The SDL is the standard that Microsoft products have been held to since 2004.



Many requirements in the SDL apply to the phases in the development process—design, implementation, verification, release, and response. Numerous tools are used to support the SDL. Some of the tools are available externally, such as the Microsoft SDL Threat Modeling Tool or Static Code Analysis (**/analyze**) in Microsoft Visual Studio® 2005 and later versions.

As of 2004, the SDL requirements became part of the release policy for Microsoft products with meaningful security risk, and each of these goes through a final security review to ensure that it complies with the SDL.

With a few minor exceptions, Silverlight met all of the SDL requirements. Some of the most important of those requirements are described below.

**Note that not all tasks below are SDL *requirements*. Some are classified as *recommendations*, but they are listed by phase for ease of reading. The SDL recommendations are labeled as optional*.***

**Training Phase**

***Security Training***

The SDL requires that at least 80 percent of product team members working in technical roles—developers, quality assurance (QA), and program managers—receive security training within the previous year. That requirement helps ensure that a product team is up to date with general security trends and helps them recognize security issues that they encounter. The Silverlight team met these training requirements.

**Requirements Phase**

***Analysis of Publicly Known Security Vulnerabilities Encountered
by Competitor Products (Optional)***

The Silverlight team analyzed competing products that were being designed for and exposed to an environment similar to the one in which Silverlight operates. This served as an excellent source of data for the kind of attacks the team might expect to see. Security vulnerabilities that have been publicly disclosed since 2000 were carefully analyzed. After triaging a select subset, a number of security improvements were recommended. Those were applied through threat model reviews, vulnerability reviews, and code changes.

**Design Phase**

***Threat Modeling***

Threat modeling is a process of analyzing an application from the standpoint of an attacker. Though this sounds simple, the process, if applied thoroughly, allows many security issues to be caught early in the design phase, before coding starts. For Silverlight 1.0, threat modeling identified a few issues, but most were minor. The issues resulted in extra testing, user documentation changes, and additional items for the code review list. No significant security design deficiencies were identified during the threat modeling process. Teams must ensure that they understand and list all possible entry points, protected resources, external dependencies, and data flow paths for the application. Based on that data, an analysis is conducted to identify potential ways to break an application. Often, design or code changes are made based on the results of threat modeling. Microsoft offers a free, downloadable [SDL threat modeling tool](http://msdn.microsoft.com/en-us/security/dd206731.aspx).

***Cross-Site Scripting Analysis (Optional)***

Cross-site scripting (XSS) security vulnerabilities are found in a large number of today’s Web applications. Although we cannot guarantee that the improper use of Silverlight will not result in an XSS vulnerability, a design-level effort was made to make sure that the presence of Silverlight on the page does not make any existing XSS vulnerabilities worse or easier to exploit.

**Implementation Phase**

***Static Code Analysis***

Static code analysis includes running several tools that analyze the product source code and identify patterns that are known to be problems from a security perspective. While no tool can replace human attention, static code analysis is generally useful for finding certain kinds of security vulnerabilities, and it complements manual code reviews. Some of the Microsoft SDL tools used for static code analysis (namely **/analyze** and FxCop) are publicly available in some editions of Visual Studio 2005 and later versions.

***Compiler Protections***

The Microsoft Visual C++® build tools support a number of runtime defense-in-depth mitigations against buffer overflows. Often, they are known by the names of corresponding compiler/linker switches, as follows:

* **/GS** helps protect function return address and some function parameters (such as pointers) by placing a security cookie in the function stack before the local variables area and verifying the cookie's state before returning from the function.
* Safe Exception Handling (**/SafeSEH**) limits the set of potentially executable exception handlers to only those registered at compile time, preventing execution flow transfer to an exception handler that is overwritten by an attacker.
* Data Execution Prevention (DEP, **/NXCOMPAT**) prevents running any code from process memory pages that are not explicitly marked for execution.
* Address space layout randomization (ASLR, **/dynamicbase**) opts in an image to be loaded at a randomly re-based location in memory, making it more difficult for an attacker to predict the addresses of key elements within the image.

These mitigations are available in the 2008 set of Microsoft build tools. They are beneficial for security in virtually any C or C++ application. Although none of these protection mechanisms is certain to protect in every situation, their use makes exploiting buffer overflows much harder for potential attackers.

All of these mechanisms are turned on in the Windows code of Silverlight.

In addition, other Microsoft build tools help to verify that all product binaries are built with the recent secure versions of compilers/linkers and that they use all proper runtime defense mechanisms, such as **/GS** stack protection, safe exception handling, and DEP.

***SAL Annotations (Optional)***

Standard Annotation Language (SAL), a technology from Microsoft Research, is a set of annotations for C++ code that helps to identify more vulnerabilities by richly annotating program source code in a way that is understood by code analysis tools. As of 2007, use of SAL was an SDL recommendation that some teams across Microsoft choose to follow. For Silverlight 1.0, all code that is exposed to untrusted data was SAL-annotated. This process detected several vulnerabilities that were fixed before the product was released.

**Verification Phase**

***Fuzz Testing***

Fuzz testing is a technique used to test program functionality and error handling, by feeding data that is deliberately malformed, corrupted, or random into a target application. Fuzz testing is especially effective in discovering buffer overflows in C/C++ code that processes untrusted input, such as media codecs that display content received over the Internet.

The SDL requires data parsers to be fuzz tested before product release. The Silverlight team not only met this requirement but also exceeded it—by 300 times. The team used several fuzzing techniques and tools, including tools both internal and external to Microsoft.

***COM Hygiene Testing***

The Silverlight plug-in is an ActiveX control; in other words, it’s a COM object. A very common mistake among COM developers is to test their COM objects with one container only (for example, Internet Explorer), hiding errors that would probably have been identified if published design and development guidance for COM objects had been followed. Many of these errors result in memory corruption, which may result in security vulnerabilities.

One vulnerability of this kind was reported during the beta release of Silverlight 1.0. As a result, an internally developed tool was applied to Silverlight to exercise and test all program interfaces. The internal tool forces all COM rules on a COM object, making sure the object could be instantiated and used within any non-malicious COM container without errors. As a result, all Silverlight COM code was clean from known types of errors of this kind when it was released.

***Security Vulnerability Reviews***

The SDL calls for monitoring security-related vulnerabilities in software and ensuring that each vulnerability is correctly resolved. Typically, that monitoring process translates into several reviews to make sure that none of the issues identified is accidentally underestimated and left unfixed.

***Directed Code Reviews (Optional)***

During directed reviews, the Silverlight team created a list of areas that might affect security. Based on that list, the team conducted several developer code review sessions to look for specific patterns in the code that could create problems. This effort was quite effective and uncovered several issues that were subsequently fixed.

***External Penetration Testing (Optional)***

Two groups of external security penetration testers were invited to Microsoft to work with the pre-release Silverlight product. Although it was important to find as many security vulnerabilities as possible, the primary goal of the penetration test was to obtain an independent evaluation of the product’s security. By subjecting the product to external testing and audit, the team could gauge whether the security work on Silverlight had met the team’s expectations. Finding too many vulnerabilities or signs of vulnerabilities concentrated in certain areas would have indicated a general weakness and could possibly have resulted in product ship cycle delay. However, the external vendors' evaluation of Silverlight 1.0 was quite positive.

**Response Phase**

Software security is complex. Even the best efforts and practices cannot guarantee that security vulnerabilities do not exist in the product after it is released. Fully realizing that, Microsoft requires a security response plan for each product, as part of the SDL. The plan requires having several people available 24 hours a day, 7 days a week, as a first line of response. The plan outlines the key procedures for incident investigation and update and establishes contacts with the Microsoft Security Response Center (MSRC). All of the response phase activities were completed for the Silverlight release.

**SPECIFIC SECURITY ACTIVITIES BEYOND THE SDL**

Despite the long history of improvements, the SDL was not readily applicable to some security challenges introduced by Silverlight. Each of these unique challenges was reviewed during several brainstorming sessions and workshops to identify threats. Then, the Silverlight team identified experts on each threat and assigned them to specific areas. With their help, a set of actionable items, described in the following subsections, was identified and followed throughout the Silverlight 1.0 release.

**Exposure in Non-Internet Explorer Browsers**

Silverlight is supported in Internet Explorer, Firefox, and Safari. Although it was relatively easy to find expertise at Microsoft for Internet Explorer, expertise with Firefox and Safari was more difficult to find, so we took the following steps:

* Based on Internet Explorer coverage, we identified missing knowledge and questions about other browsers.
* We created a board of experts with knowledge about browsers other than Internet Explorer, both within Microsoft and outside of the company, among penetration testing vendors and in the security research community.
* We combined the resulting knowledge and applied it to Silverlight 1.0 and subsequent releases, in design and code development, in addition to testing and verification.

**Cross-Platform Capabilities**

Silverlight 1.0 runs on both the Windows operating system and on Macintosh. However, the set of APIs, security tools available, and even security exposures are significantly different for these two platforms. Measures that are easily applied in Windows do not always work well on Mac OS X and vice versa.

For design and security testing (such as fuzzing), Microsoft could draw on its past experience with shipping Microsoft Office 2008 for Mac. However, static code analysis became an issue due to licensing limitations and because tools for the Mac OS were not available.

After careful analysis, we found that only a tiny fraction of Silverlight code is Macintosh-specific; the rest is platform-agnostic. As a result, common Windows-based secure coding practices and static analysis tools have been applied uniformly to the majority of the code.

After Windows testing, Mac-specific sections of source code received two independent security code reviews to ensure their quality.

**Cross-Group Collaboration**

Although cross-group collaboration is not a technical requirement of the SDL, there was significant cooperation among many groups inside Microsoft in the design, development, and testing of Silverlight 1.0. Dozens of experts across a variety of teams contributed their time and knowledge to make sure that the security quality of Silverlight is as high as it can be.

**RESULTS**

As of January 2009 (16 months after the release), Silverlight 1.0 had no public security vulnerabilities reported against it.

This result does not imply that no vulnerabilities will be found. They happen in software, and a smart, determined attacker can sometimes find them, despite the best efforts of software vendors.

However, the process used with Silverlight demonstrates that a consistent application of the SDL, along with additional best practices (which may be unique to the individual product), can reduce the number of security issues. As a result, the two efforts combined can help build more secure software that’s easier to use. Ultimately, it can even help to make the Internet safer.

**ACKNOWLEDGEMENTS**

The author wants to express his gratitude to the Silverlight team and to the Microsoft Security Engineering Center (MSEC) PM for their contribution to this paper, especially to Mark Alcazar, Jesse Collins, Marco Matos, Reid Borsuk, Christopher Swenson, Mark Cartwright, and Vishal Kumar.