The Trustworthy Computing Security Development Lifecycle

The Microsoft SDL Team
Microsoft Corporation

1. Introduction

It is imperative that all software vendors address security and privacy threats. Security is a core requirement for software vendors, driven by market forces, the need to protect critical infrastructures, and the need to build and preserve widespread trust in computing. A major challenge for all software vendors is to create more secure software that requires less updating through patches and less burdensome security management.

For the software industry, the key to meeting today's demand for improved security is to implement repeatable processes that reliably deliver measurably improved security. Therefore, software vendors must transition to a more stringent software development process that focuses, to a greater extent, on security. Such a process is intended to minimize the number of security vulnerabilities extant in the design, coding, and documentation and to detect and remove those vulnerabilities as early in the development lifecycle as possible. More importantly, an improved process must reduce the chance that new security vulnerabilities are added to software products in the first place. The need for such a process is greatest for enterprise, governmentconsumer software that is likely to be used to process inputs received from the Internet, to control critical systems likely to be attacked, or to process personally identifiable information.

There are three facets to building more secure software: repeatable process, engineer education, and metrics and accountability. This document focuses on the repeatable process aspect of the Security Development Lifecycle (SDL), although it does discuss engineer education and provide some overall metrics that show the impact to date of application of a subset of the SDL.

The SDL has two distinct goals, the first is to reduce the number of vulnerabilities in the software, and the second is to reduce the severity of security vulnerabilities that are missed. We will explain how these goals are achieved throughout this document,

If Microsoft's experience is a guide, adoption of a set of security-related development process improvements such as those defined in the SDL by other organizations should not add unreasonable costs to software development. In Microsoft's experience, the benefits of providing more secure software (e.g., fewer compromises, fewer patches, more satisfied customers) far outweigh the costs.

The SDL involves modifying a software development organization's processes by integrating measures that lead to improved software security. This document summarizes those measures and describes the way that they are integrated into a typical software development lifecycle. The intention of these modifications is not to totally overhaul the process, but rather to add well-defined security checkpoints and security deliverables.

This document outlines, at a very high level, the SDL stages shown in Figure 1.

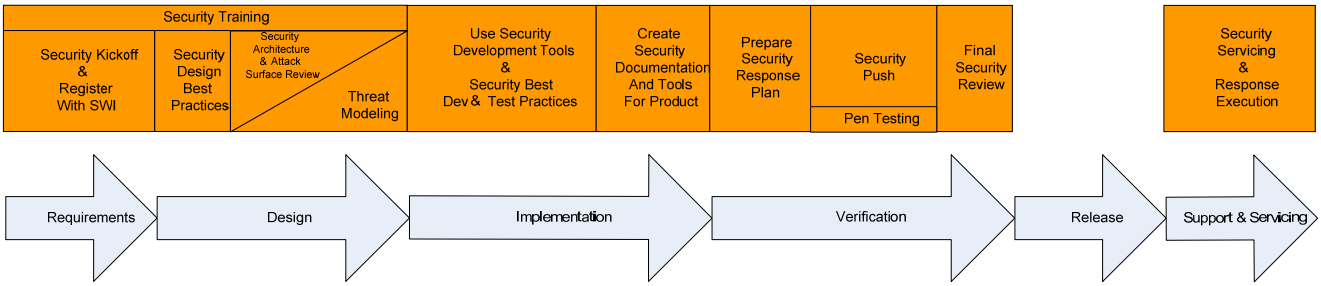

Figure 1: SDL Improvements to the Microsoft development process.

2. The Security Development Lifecycle Process

An organization that seeks to develop secure software must take responsibility for ensuring that its engineering population is appropriately educated. At Microsoft, all personnel involved in developing software must go through security training. These classes include:

- Security basics
- Secure design principles
- Threat modeling in depth
- Common vulnerabilities in depth
- Domain-specific vulnerabilities in depth
- Security testing
- Etc.

2.1 Requirements Phase

The need to consider security "from the ground up" is a fundamental tenet of secure system development. During the requirements phase, the product team makes contact with the central security team to request the assignment of a security advisor who serves as point of contact, resource, and guide as planning proceeds. The security advisor assists the product team by reviewing plans, making recommendations, and ensuring that the security team provides appropriate resources to support the product team's schedule. The security advisor remains the product team's point of contact with the security team from project inception through completion of the Final Security Review and software release.

The requirements phase is the opportunity for the product team to consider how optimally security will be integrated into the development process, identify key security objectives, and otherwise maximize software security while minimizing disruption to customer usability, plans and schedules. As part of this process, the team needs to consider how the security features and assurance measures of its software will integrate with other software likely to be used together with its software.

2.2 Design Phase

The design phase identifies the overall requirements and structure for the software. From a security perspective, the key elements of the design phase are:

2.2.1 Define security architecture and design guidelines

Define the overall structure of the software from a security perspective, and identify those components whose correct functioning is essential to system security. It is important that software be designed and built using the best security designs possible within the constraints of usability, reliability, supportability and cost. Specifics of individual elements of the architecture will be detailed in individual design specifications, but the security architecture identifies an overall perspective on security design.

2.2.2 Document the elements of the software attack surface

Given that software will not achieve perfect security, it is important that only features that will be used by the vast majority of users be exposed to all users by default, and that those features be installed with the minimum feasible level of privilege.

Attack surface measurement artifacts include:

- Open TCP ports
- Open UDP ports
- System services (daemons) running by default
- Processes (such as services, daemons or setuid applications) running in elevated identity
- Elevated user accounts (root, administrator)
- Extensible software mechanisms (ActiveX controls, Java applets, browser helper objects)

Measuring the elements of attack surface provides the product team with an ongoing metric for default security and enables them to detect instances where the software has been made more susceptible to attack. While some instances of increased attack surface may be justified by enhanced product function or usability, it is important to detect and question each such instance during design and implementation so as to ship software in as secure a default configuration as feasible.

2.2.3 Conduct threat modeling

Using a structured methodology, the threat modeling process identifies threats that can do harm to system assets and the likelihood of harm being done (an estimate of risk), and helps identify countermeasures that mitigate or reduce the risk. Threat modeling helps the product team identify the need for defensive security features as well as areas where especially careful code review and security testing are required. The threat modeling process should be supported by a tool that captures threat models in machine-readable form for storage and updating, and if possible, provide guidance for a development team.

2.3 Implementation Phase

During the implementation phase, the product team codes, tests, and integrates the software. Steps taken to remove security flaws or prevent their initial insertion significantly reduces the likelihood that security vulnerabilities will make their way into the final version of the software.

The elements of the SDL that apply are:

2.3.1 From the threat model task, understand which are the high-risk components.

This requirement helps drive many decisions made during the rest of the implementation phase. Understanding which components are at highest risks means that those components should get more security attention such as deeper code analysis, penetration and fuzz testing.

2.3.2 Apply secure coding and security testing standards.

Coding standards help developers avoid introducing flaws that can lead to security vulnerabilities. For example, the use of safer and more consistent string-handling and buffer manipulation constructs can help to avoid the introduction of buffer overrun vulnerabilities. Testing standards and best practices help to ensure that testing focuses on detecting potential security vulnerabilities rather than concentrating only on correct operation of software functions and features

2.3.3 Apply security testing tools including fuzzing tools.

"Fuzzing" supplies structured but invalid inputs to data entry points such as software application programming interfaces (APIs) and network interfaces so as to maximize the likelihood of detecting errors that may lead to software vulnerabilities.

2.3.4 Apply static-analysis code scanning tools and black-box analysis tools.

Tools can detect some kinds of coding flaws that result in vulnerabilities, including buffer overruns, integer overruns, and uninitialized variables. Microsoft has made a major investment in the development of such tools (the two that have been in longest use are known as PREfix and PREfast) and continually enhances those tools as new kinds of coding flaws and software vulnerabilities are discovered. The PREfast tool is available in Visual C++ 2005 through the /analyze compiler switch.

2.3.5 Conduct security code reviews.

Code reviews supplement automated tools and tests by applying the efforts of trained developers to examine source code and detect and remove potential security vulnerabilities. They are a crucial step in the process of removing security vulnerabilities from software during the development process.

2.4 Verification Phase

The verification phase is the point at which the software is functionally complete and enters beta testing. Microsoft introduced the security push during the verification phase of Windows Server 2003 and several other software versions in early 2002. The main purpose of the security push is to review both code that was developed or updated during the implementation phase and "legacy code" that was not modified.

There were two reasons for introducing the security push into the process:

- The software lifecycle for the versions in question had reached the verification phase, and this phase was an appropriate point at which to conduct the focused code reviews and testing required.

- Conducting the security push during the verification phase ensures that code review and testing target the finished version of the software, and provides an opportunity to review both code that was developed or updated during the implementation phase and "legacy code" that was not modified.

The first of these reasons reflects a historical accident: the decision to launch a security push initially occurred during the verification phase. But Microsoft has concluded that conducting a security push during the verification phase is actually good practice, both to ensure that the final software meets requirements and to allow deeper review of any legacy code that has been brought forward from prior software versions.

It is important to note that code reviews and testing of high priority code (code that is part of the "attack surface" for the software) are critical to several parts of the SDL. For example, such reviews and testing should are required in the implementation phase to permit early correction of any problems and identification and correction of the source of such problems. They are also critical in the verification phase when the product is close to completion.

We believe the need for a security push diminishes over time as security becomes "baked into" the software product. For example, there was no security push for Windows Vista, because Windows Server 2003 and Windows XP SP2 had been through the early incarnations of the SDL.

2.5 Release Phase

During the release phase, the software should be subject to a Final Security Review ("FSR").

The FSR is an independent review of the software conducted by the central security team for the organization. Tasks include reviewing bugs that were initially identified as security bugs, but on further analysis were determined not to have impact on security, to ensure that the analysis was done correctly. An FSR also includes a review of the software's ability to withstand newly reported vulnerabilities affecting similar software. An FSR for a major software version will require penetration testing and, potentially, the use of outside security review contractors to supplement the security team.

For some of the larger products, Microsoft employs external security consulting organizations to help perform the FSR. These companies are hired based on their areas of expertise and skillsets. In general, these organizations find security issues that could only be found by individuals or groups with a great deal of skill. Very few, if any, tools can find such vulnerabilities.

2.6 Support and Servicing Phase

Despite the application of the SDL during development, state of the art development practices do not yet support shipping software that is completely free from vulnerabilities – and there are good reasons to believe that they will never do so, most notably the area of security research is constantly evolving:software security is humans pitted against humans. Product teams must prepare to respond to newly-discovered vulnerabilities in shipping software.

Part of the response process involves preparing to evaluate reports of vulnerabilities and release security advisories and updates when appropriate. The other component of the response process is conducting a post-mortem of each reported vulnerability and taking action as necessary.

3. Results of Implementing the Security Development Lifecycle at Microsoft

Since the inception of the SDL at Microsoft, the company has seen a decline in both the quantity and severity of security vulnerabilities that affect customers. At a very high level, the quantity of security vulnerabilities has dropped by approximately 50%, and some products have show much greater progress. For example, since its release in 2003 Microsoft's Web server software, Internet Information Services 6.0 has had only three security bulletins. More importantly, the bulletins apply to features that are not enabled by default. Another example is SQL Server 2005. Microsoft has issued no security bulletins for the SQL Server 2005 database engine in three years, again, this is a testament to the work performed by the development teams as they adhere to and apply the SDL.

The first version of Windows to come through a complete SDL cycle is Windows Vista, and while it is too early to measure true progress, after nine months it appears that Windows Vista is showing an approximate 50% reduction in security vulnerabilities compared to Windows XP SP2.

Finally, the first version of Microsoft Office to come through the SDL is Microsoft Office 2007. Over 90% of the recent file format vulnerabilities in older versions of Office do not affect Microsoft Office System 2007, this can be attributed to two SDL requirements: Integer overflow analysis requirements and fuzzing requirements.

Finally, we would like to say that while a 50% drop in security vulnerabilities does not equate to zero security vulnerabilities, a 50% reduction is considerable progress.

4. Conclusions

Microsoft's experience indicates that the SDL is effective at reducing the incidence of security vulnerabilities. The company's customers have seen a slow and steady reduction in vulnerabilities affecting newer versions of all products developed using the SDL.

The process is not perfect, and is still evolving – and is unlikely either to reach perfection or to cease evolving in the foreseeable future.

The development and implementation of the Security Development Lifecycle represent a major investment for Microsoft, and a major change in the way that software is designed, developed, and tested. The increasing importance of software to society emphasizes the need for Microsoft and the industry as whole to continue to improve software security; therefore, both this paper on the SDL and books on specific techniques have been published in an effort to share Microsoft's experience across the software industry.

5. References

Lipner, Steve and Michael Howard, "The Security Development Lifecycle" Microsoft Press 2006

6. Notices