
The Security Development Lifecycle (SDL). Advantage, Microsoft

Date: November, 2006

Author: Jon Oltsik Senior Analyst

Abstract: When it comes to Microsoft and security, few people ever mention Microsoft's Security Development Lifecycle (SDL). ESG believes this is an unfortunate omission. The fact is that Microsoft's commitment to SDL is an area of stealthy security leadership. ESG believes that other ISVs should embrace an SDL model as soon as possible and that enterprise organizations should mandate that technology vendors establish a measurable and transparent SDL process by 2008 or risk losing business.

A basic formula for Risk Management is as follows:

$$\text{Threat} + \text{Vulnerability} = \text{Risk}$$

In this equation, a threat is defined as "any natural or man-made event that could have an adverse or detrimental impact," and a vulnerability is defined as "the absence or weakness of a safeguard in an asset that makes a threat potentially more harmful or costly." Within this formula, threats are relatively fixed and constant. If you live in Florida for several years, you will likely experience a hurricane. Since threats are ubiquitous and uncontrollable, the key to security is risk mitigation - address vulnerabilities and thus reduce risk to an acceptable level.

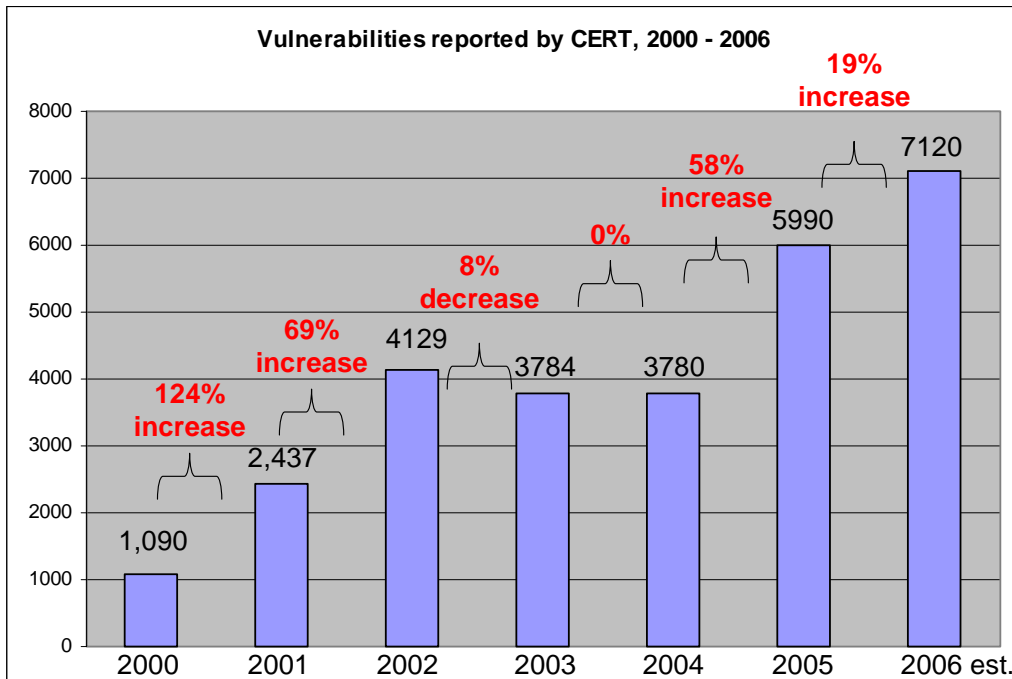
When risk management principles are applied to software development, mitigation techniques seem obvious - lower the number of vulnerabilities and/or reduce the severity of vulnerabilities in the code and you've effectively lowered your risk. In fact, simple statements like "write better code," have become the mantra of the security world for years. Unfortunately, this obvious solution is easier said than done. Why? First, developers and ISVs must get their heads out of the sand and admit that there is a problem here. Sadly, even when this happens - and it will -- developing secure code still remains elusive because:

- **Features and functions pay the bills.** Let's face it, secure coding takes a seat in the back of the bus when weighed against developing a new software package for driving new revenue, automating business processes, or lowering transaction costs. Under tight deadlines, software developers are motivated to get buggy code out the door and fix security and quality problems as they arise.
- **Coders lack security-focused development skills.** Even if the software development team were mandated to adopt security best practices, few would have a clue where to start. Leading technical schools such as Carnegie Mellon, MIT, Stanford and UC-Berkeley are either just beginning to offer courses on secure development practices or treat security as one of several elements related to software "quality." This means that a seasoned software engineer with 10 years of experience has almost no secure development knowledge at all.
- **Software tools only find what they are looking for.** Some developers view the new crop of security testing tools as a panacea. Yes, source code crawlers, vulnerability scanners and fuzz testers are great ways to help scale security testing and computers don't get tired or distracted when performing long tedious processes. That said, testing tools are prone to false positives and won't catch well

written code that also happens to introduce a security risk. A tool that scans source code for buffer and heap overflows won't catch a faulty logic statement, a poor design or an unnecessary interface.

In an era where technology is woven into business processes, this paints a rather dire picture. Little wonder that the number of annual software vulnerabilities (for all software, not just Microsoft) reported by the CERT Coordination Center continues to rise (see Figure One).

Figure One: Cert/CC Statistics for Software Vulnerabilities Reported, 2000 - 2006



Source: Cert/CC Statistics

Microsoft and the Birth of the Security Development Lifecycle (SDL)

Developing secure software is analogous to losing weight. There are plenty of products out there promising great benefits but the only proven way to effectively shed pounds is through behavioral modifications including dietary changes and increased exercise. The same holds true for secure software development. Miracle tools may promise great results but the only surefire way to produce secure code is to integrate security into the development process itself. In an environment of more frequent and sophisticated attacks, an SDL has morphed from a "nice to have" to a "gotta have."

While all ISVs pay lip service to secure software development, Microsoft has become a demonstrable leader in this area. Most people equate Microsoft's security effort with the now famous "Trustworthy Computing" e-mail manifesto delivered to all Microsoft employees in early 2002. In reality, security-focused process improvements were well under way by then.

As far back as 1998, the roots of SDL were already firmly in place when Microsoft assembled its Internal Security Task Force - a group of individual specialists who worked on security requirements with various product development teams. In the Windows 2000 timeframe, security requirements became more integrated into the development process and the Secure Windows Initiative (SWI) was born. This added security-focused design and code reviews, new forms of security testing and also gave the security team the ability to "stop shipment" if the product was still deemed to contain any high risk vulnerabilities. Over the years,

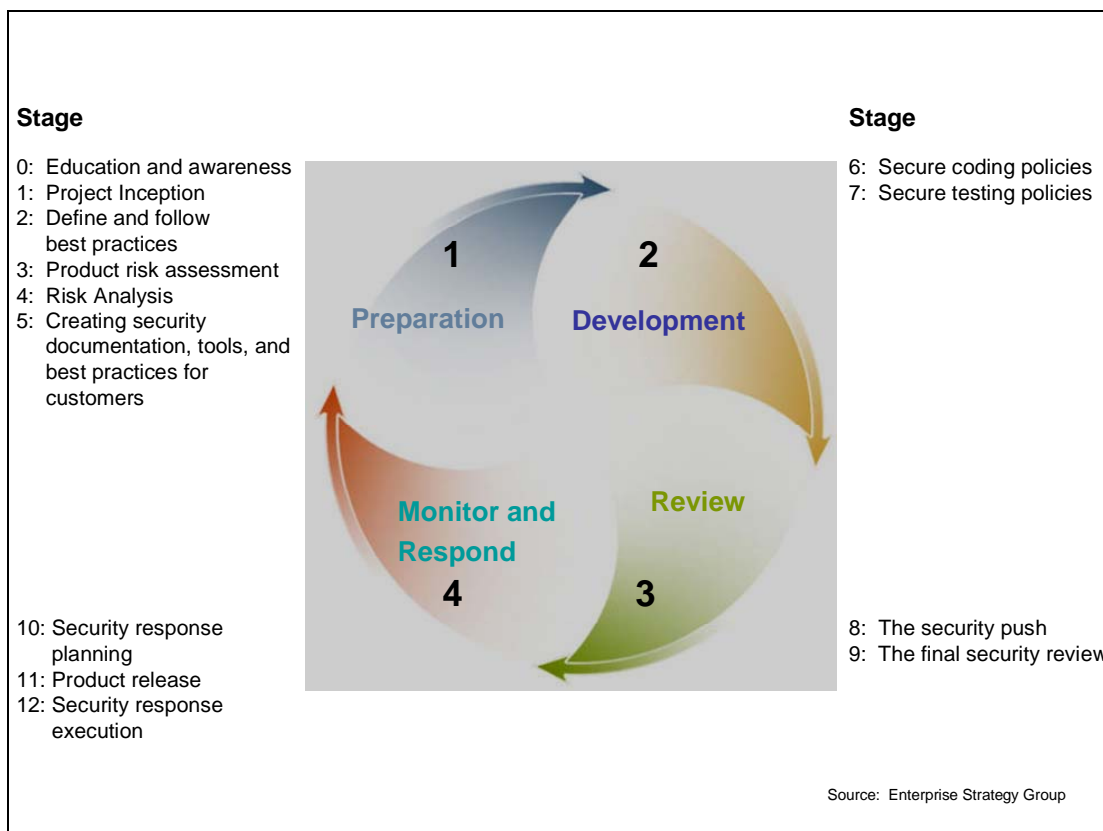
security-focused development processes continued to mature as it added more emphasis on up-front training, attack surface analysis (ASA), attack surface reduction (ASR) and secure “by default” configurations.

From 2000 through 2004, new product releases would go through an increasingly sophisticated “security push.” Security quality was steadily improving but the security push was still based on many ad-hoc processes managed by a growing number of knowledge experts. Given the increasing need for secure code, the SDL got its biggest boost when Microsoft’s senior management decided to institutionalize secure software development with the formalized definition of the SDL. SDL architects Michael Howard and Steve Lipner believe that the commitment from executive management was a turning point that transformed SDL from an adjunct to the development process to a fundamental component of the Microsoft culture. From that point on, all software used for business activities, containing personal/sensitive information or used on the Internet would be subject to the SDL process.

The SDL in Simple Terms

The SDL is made up of 13 Stages (Stage 0 through 12) that start with education and awareness and proceed through security response execution. A thorough analysis of these stages is beyond the scope of this brief, but ESG believes that the 13 steps can be summarized with a simplified lifecycle model as follows (see Figure Two):

Figure Two: Simplified Illustration of the SDL Process



1. **Preparation.** Stages 0-5 include defined processes for training, best practices, risk analysis and customer-centric security documents, tools and procedures. In this way, the SDL is intended to make developers think through security from a project development, planning and customer requirement perspective. The goal here is to prepare effectively upfront, eliminate surprises and extend security to all customer deliverables, not just the binaries.

2. **Development.** Stages 6-7 lay out the recipe for secure coding and testing. Secure coding depends upon earlier steps, including training and best practices. SDL also emphasizes that known dangerous functionality be replaced with a safer alternative. Secure testing is an especially mysterious topic. SDL emphasizes fuzz testing (i.e., using malformed data and packets to test application and protocol behavior) and penetration testing. It is also important to re-review product design and attack surfaces in this stage to limit potential vulnerabilities.
3. **Review.** Stages 8-9 focus on the security push and final review. This is where everything is checked from a security perspective including the code, documentation and all the milestones of the SDL process. If the SDL has been followed strictly, final product security review should be relatively straightforward with known vulnerabilities reasonably low and further risks thoroughly understood.
4. **Monitor and respond.** Even with rigorous adherence to the SDL, new security vulnerabilities are inevitable. To deal with this, stages 10-12 of the SDL concentrate on response planning, product release and response execution. During these stages, it is critical to analyze field data, rapidly test potential problems and prepare various levels of responses. Technical responses must be supported with strong communications so that customers clearly understand, assess, and react to their own risks.

In spite of its rigor, the SDL methodology remains a dynamic “work-in-progress” under constant review for improvement. Microsoft opens SDL for review and process changes twice each year in order to accommodate process improvements and evolving security threats. Like Six Sigma and ISO standards, Microsoft believes that the SDL’s commitment toward constant improvement is critical to its ongoing success.

SDL Costs and Benefits

Microsoft makes no bones about it: the SDL is not easy or free. In fact, SDL authors Michael Howard and Steve Lipner estimate that the SDL can add 15% to 20% more time to a design cycle containing significant legacy code.

All things being equal, is it really worth it? Microsoft thinks so. Howard and Lipner enthusiastically point to metrics such as a 50% decrease in vulnerabilities in all software products, like SQL Server 5.0, that have gone through the SDL from start to finish. SDL guru Michael Howard thinks of SQL Server as an SDL “poster child.” As of this writing, SQL Server has not exhibited a security vulnerability in the actual database engine in over 3 years. The upcoming Vista release will be the first version of Windows to go through the process and Microsoft expects similar results. Yes, Vista will have vulnerabilities like every other piece of sophisticated software, but Microsoft believes that over the next few years, Vista will demonstrate an overall reduction in vulnerabilities over past Windows versions as well as a reduction in the severity of those vulnerabilities that are found. In general terms, Microsoft product security will continue to improve as other products pass through the SDL gate for the first time and get better through subsequent trips.

Secure software development also fits neatly into the idiomatic expression, “an ounce of prevention is worth a pound of cure.” Several academic papers conclude that:

- One hour of software QA activities can save between 3 and 10 hours of post-release remediation work.
- An undetected software requirement defect can cost 50 to 200 times as much to fix when discovered later in the development or post-development process.
- A defect found and fixed during a code review would cost 10 to 100 times as much to fix when discovered later in the development or post-development process.

Microsoft also points to the cost savings of SDL as it relates to the “multiplier effect.” It is far cheaper to fix or prevent a vulnerability in the development cycle than to patch 100 million+ desktops in the field!

Given these metrics, it is clear that the SDL will more than pay for itself over the long-term in development cost alone. This combined with improved reputation, higher quality software and increasing customer satisfaction make the SDL a “no brainer.” Little wonder that Microsoft is sharing SDL tools with partners. As strong as Redmond is at marketing, SDL is one area where it is too humble. ESG believes that Microsoft ought to be making SDL a competitive differentiator in every large enterprise software deal.

The Bottom Line

Many technology professionals and vendors continue to pooh-poo Microsoft when it comes to processes and software quality. ESG believes that this couldn't be farther from reality. The SDL is a visible example that Microsoft recognizes the need for software development process re-engineering and the implementation a real solution. As such, Microsoft has once again trumped the market.

Microsoft's effort should be applauded, but Redmond is not the only ISV in town. This begs the question: What are other software vendors and technology companies doing to improve the security of their code? ESG believes that the answer is “not enough.”

In order to drive an industry-wide effort to improve software security, enterprise organizations should mandate that their technology vendors adopt an SDL similar to Microsoft's (Note that the current PCI specification [Version 1.1] recommends security code reviews as a current best practice and will make this a requirement in 2008). The SDL process must be documented and large customers should have the opportunity to review the process and even audit the results.

With the PCI specification as a template, ESG believes that by 2008, all enterprise RFIs and RFPs should require that vendors provide SDL processes and metrics. Those that are unwilling or unable to do so should be politely shown the door.