The Trustworthy Computing Security Development Lifecycle

Steve Lipner
Security Engineering and Communications
Security Business and Technology Unit
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

Abstract

This paper discusses the Trustworthy Computing Security Development Lifecycle (or simply the SDL), a process that Microsoft has adopted for the development of software that needs to withstand malicious attack. The process encompasses the addition of a series of security-focused activities and deliverables to each of the phases of Microsoft's software development process. These activities and deliverables include the development of threat models during software design, the use of static analysis code-scanning tools during implementation, and the conduct of code reviews and security testing during a focused "security push". Before software subject to the SDL can be released, it must undergo a Final Security Review by a team independent from its development group. When compared to software that has not been subject to the SDL, software that has undergone the SDL has experienced a significantly reduced rate of external discovery of security vulnerabilities. This paper describes the SDL and discusses experience with its implementation across a range of Microsoft software.

1. Introduction

It is imperative that all software vendors address security 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

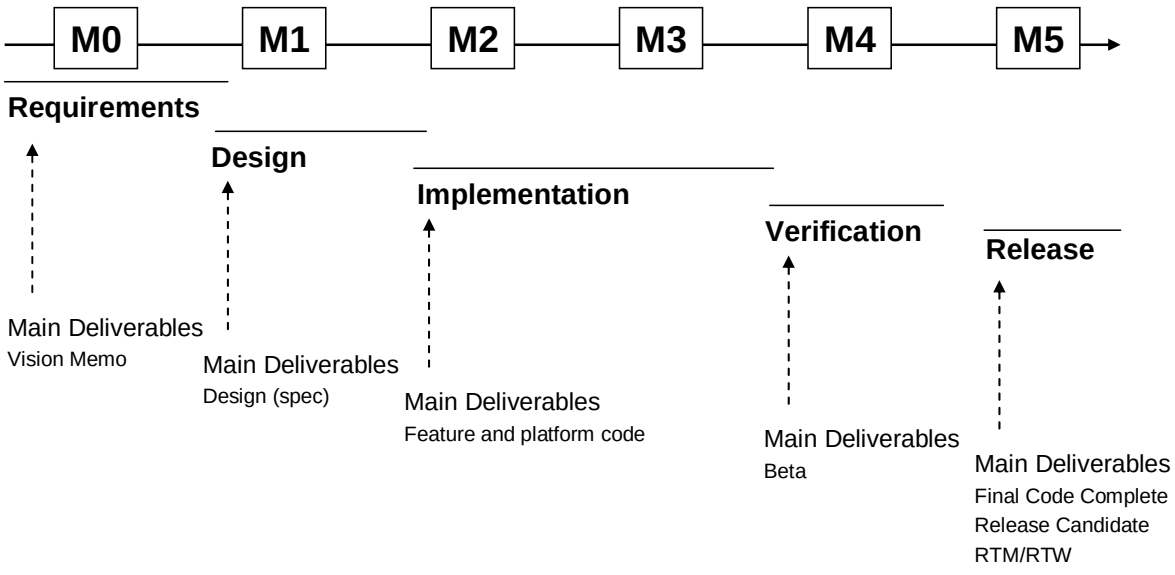

Figure 1. Baseline development process

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. The need for such a process is greatest for enterprise and consumer 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 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.

If Microsoft's experience is a guide, adoption of 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 patches, more satisfied customers) 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 assumes that there is a central group within the company (or software development organization) that drives the development and evolution of security best practices and process improvements, serves as a source of expertise for the organization as a whole, and performs a review (the Final Security Review or FSR) before software is released. In Microsoft's experience, the existence of such an organization is critical to successful implementation of the SDL as well as to improving software security. While some organizations might consider having the "central security team" role performed by a contractor or consultant, This paper describes the integration of a set of steps intended to improve software security into the software development process that is typically used by large software development organizations. These steps have been designed and implemented by Microsoft as part of its Trustworthy Computing Initiative. The goal of these process improvements is to reduce the quantity and severity of security vulnerabilities in software used by customers. In this document, the modified software development process, which is currently being implemented at Microsoft, is referred to as the Trustworthy Computing Software Development Lifecycle (or simply the SDL).

Microsoft experience is that the security team must be available for frequent interactions during software design and development, and must be trusted with sensitive technical and business information. For these reasons, the preferred solution is to build a security team within the software development organization (although it may be appropriate to engage consultants to help build and train the members of the team).

1.1 The Baseline Process

The generally accepted software development process at Microsoft follows roughly the flow shown in Figure 1. At a high level, this process is typical of industry practice.

While Figure 1 shows five milestones and appears to suggest a "waterfall" development process, the process is in fact a spiral. Requirements and design are often revisited during implementation, in response to changing market needs and to realities that arise during software implementation. Furthermore, the development process emphasizes the need to have running code at almost every point, so each major milestone is in fact broken into the delivery of a series of builds that can be tested and used operationally (by the development team) on an ongoing basis.

1.2 Security Development Lifecycle Overview

Experience with the security of real-world software has led to a set of high-level principles for building more secure software. Microsoft refers to these principles as SD^3+C – Secure by Design, Secure by Default, Secure in Deployment, and Communications. The brief definitions of these principles are:

- Secure by Design: the software should be architected, designed, and implemented so as to protect itself and the information it processes, and to resist attacks.

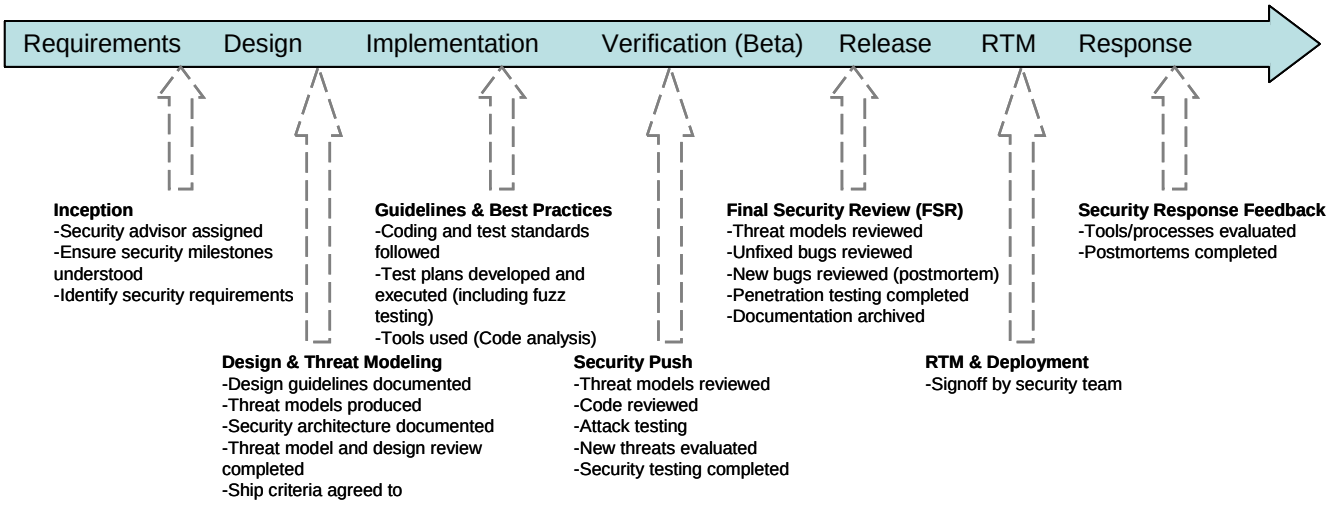

Figure 2. SDL Overview

- Secure by Default: in the real world, software will not achieve perfect security, so designers should assume that security flaws will be present. To minimize the harm that occurs when attackers target these remaining flaws, software's default state should promote security. For example, software should run with the least necessary privilege, and services and features that are not widely needed should be disabled by default.

- Secure in Deployment: software should be accompanied by tools and guidance that help end users and/or administrators use it securely. Additionally, updates should be easy to deploy.

- Communications: software developers should be prepared for the discovery of product vulnerabilities and should communicate openly and responsibly with end users and/or administrators to help them take protective action (such as patching or deploying workarounds).

While each element of SD^3+C imposes requirements on the development process, the first two elements – secure by design and secure by default – have the largest impact. Secure by design mandates processes intended to prevent the introduction of vulnerabilities in the first place, while secure by default requires that the default exposure of the software – its "attack surface" be minimized.

Introducing security measures that are intended to integrate the SD^3+C paradigm into the existing development process results in the overall process organization shown in Figure 2.

Section 2 of this document describes the components of the SDL at a high level. Section 3 presents a brief summary of the specifics of Microsoft's implementation of the SDL. Section 4 of this document provides some illustrative data that demonstrates that early application of the SDL during the development of Microsoft® Windows® Server 2003 and other software has resulted in reduced security vulnerability counts and reduced security vulnerability severity ratings compared to prior software versions. Section 5 provides some qualitative observations on elements of the process based on Microsoft's experience in the application of the SDL. Finally, Section 6 presents overall conclusions.

2. The Security Development Lifecycle Process

As noted previously, engineer education is beyond the scope of this paper. But is it important to note that an education program is critical to the success of the SDL. New college and university graduates in computer science and related disciplines generally lack the training necessary to join the workforce ready and able to design, develop, or test secure software. Even those who have completed course work in security are more likely to have encountered cryptographic algorithms or access control models than buffer overruns or canonicalization flaws.

Under those circumstances, an organization that seeks to develop secure software must take responsibility for ensuring that its engineering population is appropriately educated. Specific ways of meeting this challenge will vary depending on the size of the organization and the resources available. An organization with a large engineering population may be able to commit to building

an in-house program to deliver ongoing security training to its engineers, while a smaller organization may need to rely on external training.

2.1 Requirements Phase

The need to consider security "from the ground up" is a fundamental tenet of secure system development. While many development projects produce "next versions" that build on previous releases, the requirements phase and initial planning of a new release or version offers the best opportunity to build secure software.

During the requirements phase, the product team makes contact with the central security team to request the assignment of a security advisor (referred to as the "security buddy" in the implementation of the SDL at Microsoft) 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 plans appropriate resources to support the product team's schedule. The security advisor advises the product team on the security milestones and exit criteria that will be required based on project size, complexity, and risk. 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 security advisor also serves as the contact between the security team and product team management, and advises team management whether the security element of their project is on track so as to avoid security-related surprises late in the process.

The requirements phase is the opportunity for the product team to consider how security will be integrated into the development process, identify key security objectives, and otherwise maximize software security while minimizing disruption to 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.[1] The product team's overall perspective on security goals, challenges, and plans should be reflected in the planning documents that are produced during the requirements phase. While plans are subject to change as the project proceeds, early articulation of these plans helps to ensure that no requirements are overlooked or raised as last-minute surprises.

Each product team should consider security feature requirements as part of this phase. While some security feature requirements will be identified in response to threat modeling, user requirements are likely to dictate the inclusion of security features in response to customer demand. Security feature requirements will also be raised by the need to comply with industry standards and by certification processes such as the Common Criteria. The product team should recognize and reflect these requirements as part of its normal planning process.

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:

- 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 security (the "trusted computing base"). Identify design techniques, such as layering,[2] use of strongly-typed language, application of least privilege, and minimization of attack surface, that apply to the software globally. 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.

- 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. 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

[1] Interfacing with other software is a crucial consideration for meeting users' needs to integrate individual products into secure systems.

[2] Layering refers to the organization of software into well-defined components that are structured so as to avoid circular dependencies among components – components are organized into layers and a higher layer may depend on the services of lower layers, while lower layers are forbidden from depending on higher layers.

implementation so as to ship software in as secure a default configuration as feasible.

- Conduct threat modeling. The product team conducts threat modeling at a component-by-component level. Using a structured methodology, the component team identifies the assets that the software must manage and the interfaces by which those assets can be accessed. The threat modeling process identifies threats that can do harm to each asset and the likelihood of harm being done (an estimate of risk). The component team then identifies countermeasures that mitigate the risk – either in the form of security features such as encryption, or in the form of proper functioning of the software that protects the assets from harm. Thus, threat modeling helps the product team identify needs for 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.

- Define supplemental ship criteria. While basic security ship criteria should be defined at the organization level, individual product teams or software releases may have specific criteria that must be met before software can be released. For example, a product team that is developing an updated version of software that is shipping to customers and subject to extensive attack might elect to require that its new version be free from externally-reported vulnerabilities for some period before being considered ready for release. (That is, the development process should have found and removed the vulnerabilities before they were reported rather than the product team having to "fix" them after they are reported.)

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 during this phase are highly leveraged – they significantly reduce the likelihood that security vulnerabilities will make their way into the final version of the software that is released to customers.

The results of threat modeling provide particularly important guidance during the implementation phase. Developers pay special attention to ensuring the correctness of code that mitigates high-priority threats and testers focus their testing on ensuring that such threats are in fact blocked or mitigated.

The elements of the SDL that apply in the implementation phase are:

- Apply coding and testing standards. Coding standards help developers avoid introducing flaws that can lead to security vulnerabilities. For example, the use of safer 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.

- Apply fuzzing tools. "Fuzzing" supplies structured but invalid inputs to software application programming interfaces (APIs) and network interfaces so as to maximize the likelihood of detecting errors that may lead to software vulnerabilities.

- Apply static-analysis code scanning 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.

- Conduct 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 user beta testing. During this phase, while the software is undergoing beta testing, the product team conducts a "security push" that includes security code reviews beyond those completed in

the implementation phase as well as focused security testing.

Microsoft introduced the security push during the verification phase of Windows Server 2003 and several other software versions in early 2002. 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.

2.5 Release Phase

During the release phase, the software should be subject to a Final Security Review ("FSR"). The goal of the FSR is to answer one question. "From a security viewpoint, is this software ready to deliver to customers?" The FSR is conducted two to six months prior to software completion, depending on the scope of the software. The software must be in a stable state before the FSR, with only minimal non-security changes expected prior to release.

The FSR is an independent review of the software conducted by the central security team for the organization. The security advisor from the security team advises the product team of the scope of the FSR required by the software and provides the product team with a list of resource requirements prior to the FSR. The product team provides the security team with the resources and information needed to complete the FSR. The FSR begins with completion of a questionnaire by the product team, and an interview with a security team member assigned to the FSR. Any FSR will require a review of 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.

The FSR is not simply a pass/fail exercise, nor is the objective of the FSR to find every remaining security vulnerability in the software; this would clearly be infeasible. Rather, the FSR gives the product team and the organization's top management an overall picture of the security posture of the software and the likelihood that it will be able to withstand attack after it has been released to customers. If the FSR finds a pattern of remaining vulnerabilities, the proper response is not just to fix the vulnerabilities found, but to revisit the earlier phase and take other pointed actions to address root causes (e.g., improve training, enhance tools).

2.6 Response 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. Even if the development process could eliminate every vulnerability from software as shipped, new attacks would be discovered and software that was "secure" would be found to be vulnerable. Thus, product teams must prepare to respond to newly-discovered vulnerabilities in software that is shipping to customers.

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. Actions in response to a vulnerability range

from issuing an update in response to an isolated error to updating code-scanning tools to initiating code reviews of major subsystems. The objective during the response phase is to learn from errors and to use the information provided in vulnerability reports to help detect and eliminate further vulnerabilities before they are discovered in the field and used to put customers at risk. The response process also helps the product team and security team adapt processes so similar errors are not introduced in the future.

3. Implementing the Security Development Lifecycle at Microsoft

Microsoft's implementation of the SDL has evolved since the "security pushes" of early 2002. In order to initiate the process and to impact products far into development, the security pushes compressed into a relatively short period activities that should have been distributed across multiple phases of the SDL. The security pushes have had a significant impact on product teams' plans, resources, and schedules, and would have been much more difficult to undertake without active support from Microsoft's top management. The security pushes focused on threat modeling, code reviews, and security (including penetration) testing. The Final Security Review ("FSR") was introduced in late 2002 and early 2003, before Windows Server 2003 was released, and the FSR had a significant impact on the default configuration of Windows Server 2003 as shipped.

After the initial security pushes and FSRs, Microsoft initiated a project to formalize the SDL process. Four specific results of this project are worth specific mention:

- Policy for implementing mandatory application of the SDL.
- Mandatory education of engineering personnel.
- Metrics for product teams.
- The role of the central security team.

The following sections discuss each of these areas.

3.1 Mandatory Application of the SDL

Given the demonstrated benefits of the SDL (see Section 5), Microsoft made the decision to formalize a requirement for application of the SDL across a broad range of software. As of the writing of this document, the SDL is becoming mandatory for any software that is:

- Expected to be used to process personal or sensitive information.
- Expected to be used in an enterprise or other organization (including academia, government, or non-profits).
- Expected to be connected to the Internet or otherwise used in a networked environment.

Software to which the mandate does *not* apply includes stand-alone applications which do not fit the criteria above (e.g., games for very young children, like "The Magic Schoolbus" series). Significantly, the SDL does forbid such software from interfering with the security of the platform (operating system or other software) on which the software operates.

3.2 Mandatory Education

One key aspect of the security pushes of early 2002 was product group team-wide training for all developers, testers, program managers, and documentation personnel. Microsoft has formalized a requirement for annual security education for engineers in organizations whose software is subject to the SDL. The need for an annual update is driven by the fact that security is not a static domain: threats, attacks and defenses evolve. As a result, even engineers who have been fully competent and qualified on the aspects of security that affect their software must have additional training as the threat landscape changes. For example, the importance of integer overflow vulnerabilities has increased dramatically in the last three years.

Microsoft has developed a common introduction and update on security that is presented to engineers in both "live training" and digital media form. Microsoft has used this course as the basis for specialized training by software technology and by engineer role. Microsoft is in the process of building a security education curriculum that will feature further specialization by technology, role, and level of student experience.

Many Microsoft partners and customers have asked about the availability of Microsoft's security education and processes. Microsoft Press has published books based on Microsoft's internal practices in secure design, coding, and threat modeling, and Microsoft Learning offers courses based on Microsoft's internal practices.

3.3 Metrics for Product Teams

As a company, Microsoft is driven by the adage that "you can't manage what you can't measure." While it is very difficult to devise metrics that reliably measure the security of software, there are clearly metrics that serve as proxies for software security. These metrics range from training coverage for engineering staff (at the beginning of the development lifecycle) to the rate of discovered vulnerabilities in software that has been released to customers.

Microsoft has devised a set of security metrics that product teams can use to monitor their success in implementing the SDL. These metrics address team implementation of the SDL from threat modeling through code review and security testing to the security of the software presented for FSR. As these metrics are implemented over time, they should allow teams to track their own performance (improving, level, or deteriorating) as well as their performance in comparison to other teams. Aggregate metrics will be reported to senior product team management and Microsoft Executives on a regular basis.

3.4 The Central Security Team

Well before the security pushes of 2002, Microsoft had established the Secure Windows Initiative ("SWI") team with the role of improving software security and reducing vulnerabilities in Windows, and providing security support to product teams beyond those that develop Windows. The SWI team played the central role in organizing and managing the Windows Server 2003 security push, and provided training and consulting support for all of the security push efforts conducted beginning in 2002. The SWI team also executed the FSR for Windows Server 2003, pioneering the FSR process.

With the formal rollout of the SDL, the SWI team has taken on the role of central security team for Microsoft. The responsibilities of the SWI team include:

- Development, maintenance, and enhancement of the SDL, including definition of mandatory aspects of the process.
- Development, enhancement, and delivery of engineer education.
- Provision of "security advisors" who guide product teams through the process, conduct reviews for product teams, and ensure that product team questions receive timely, accurate, and authoritative responses.
- Serving as subject matter experts on a broad range of security topics, ensuring that questions directed to or through security advisors receive timely and accurate answers.
- Execution of Final Security Reviews before software is released.
- Technical investigation of reported vulnerabilities in software that has been released to customers, to ensure that root causes are understood and the proper level of response is initiated.

The availability and effectiveness of the SWI team have proven to be key factors in implementing the SDL at Microsoft. Microsoft aims to have a scalable process for developing more secure software, and this aim implies a need to have security competence broadly distributed across all product teams. However, having a central and highly qualified security team is key to bringing product teams across the company up to speed and supporting them as they work to build more secure software. It also ensures that the FSR is conducted by someone outside of the product team.

4. Results of Implementing the Security Development Lifecycle at Microsoft

It is premature for Microsoft to make conclusive claims that the SDL improves the security of Microsoft software, but the results to date are encouraging.

Windows Server 2003 was the first operating system release at Microsoft that implemented large portions of the SDL. Figure 3 shows the number of security bulletins and the severity of each bulletin issued within the year after release for the two most recent Microsoft server operating systems:[3] Windows 2000 and Windows Server 2003. As has been discussed earlier in this paper, Windows Server 2003 was developed with most (but not all) the SDL processes; Windows 2000 was not developed with these processes.

The severity classes are defined at http://www.microsoft.com/technettechnet/security/bulletin/rating.asp.

[3] When Windows 2000 was released, Microsoft did not have a formal security bulletin severity rating system. Microsoft has evaluated each security bulletin that applies to Windows 2000 against Microsoft's current severity rating system.

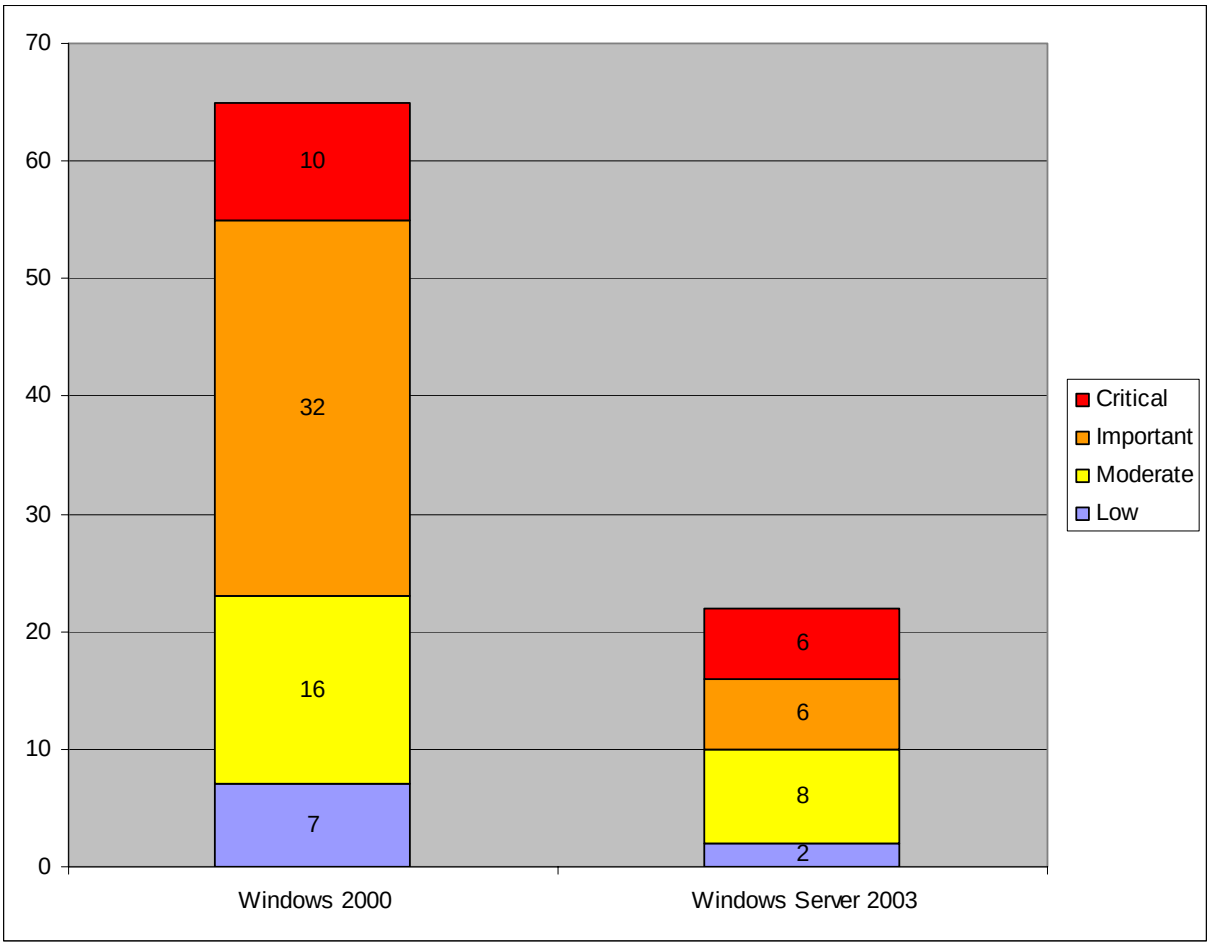

Figure 3. First Year Security Bulletins: Windows 2000 vs. Windows Server 2003

Figure 4. Server Product Security Bulletins Before and After Security Push Service Pack Release

elements of the SDL. The SQL Server and Exchange Server product teams each conducted a security push (including threat modeling, code reviews, and security testing) before releasing a service pack – a software release that aggregates fixes for both security vulnerabilities and other problems. The results of the SQL Server security push were incorporated in SQL Server 2000 Service Pack 3, and the results of the Exchange Server security push were incorporated in Exchange 2000 Server Service Pack 3. Figure 4 shows the numbers of security bulletins released in equal periods before and after the release of the respective service pack (a period of 24 months for SQL Server 2000 and 18 months for Exchange 2000 Server).

While the samples of security vulnerabilities are still small and the time periods are limited, these results provide evidence that the SDL is effective. Microsoft will continue to monitor the rates of vulnerabilities in Windows Server 2003 and the Exchange Server and SQL Server service packs to see if the early trends continue. Microsoft will also analyze other Microsoft software as new versions are shipped after full implementation of the SDL to determine if the numbers and severity ratings of security vulnerabilities continue to fall

5. Observations on Applying the Security Development Lifecycle

The data presented in the previous section provided an overview of "what" the SDL is supposed to accomplish. This section attempts to answer some questions about "how" the process works. While the previous section is based on hard numbers – Microsoft tracks vulnerability reports and security bulletins rigorously – this section is based on anecdotal data in the form of observations and opinions of people in the SWI team.

5.1 Effectiveness of Elements of the SDL

The SDL is composed of a large number of component sub-processes that are distributed throughout the software development lifecycle. The SDL team has been asked to prioritize those sub-processes in terms of effectiveness – which ones have the highest payoff, and what has been tried and been found less effective.

The consensus across the SWI team is that threat modeling is the highest-priority component of SDL. Obviously, threat modeling is not applied in isolation: threat modeling drives design, code review and testing, and a process that implemented only threat modeling but then took no action in response to the models (by failing to test the effectiveness of mitigations for example) would not be effective at all. Statistics in the form of bug counts tend to understate the role of threat modeling because much of the contribution of threat modeling is to ensure that bugs that would lead to security vulnerabilities are never created. However, the role of threat modeling in focusing the process of developing secure software is so critical that it clearly rises to the top of the list.

The SDL is still a relatively new process at Microsoft, so there have as yet been no instances in which a component of the process has been removed. However, one finding will come as no surprise to long-time security researchers: penetration testing is not the way to achieve security. Penetration testing is an element of the Final Security Review (FSR) for a major software release, but product team activities throughout the entire lifecycle focus on threat modeling, code reviews, the use of automated tools, and fuzz testing rather than penetration testing. The latter measures are much more thorough in preventing or removing security bugs than the classic ad hoc penetration testing. The penetration testing element of FSR helps to determine whether software is ready for release rather than being a way to find and fix security bugs. If the penetration test at FSR is highly productive of security bugs, it is because earlier phases have not been effective enough, and the correct response is to revisit activities that were supposed to have been completed in those phases rather than only fixing the penetration test bugs and release the software.

5.2 Tools, Testing, and Code Reviews

Static analysis tools, fuzz testing, and code review are all complementary. Microsoft has invested heavily in static-analysis code scanning tools, and the use of these tools is an integral part of the SDL. The tools are effective in finding many coding errors that can lead to security vulnerabilities – especially buffer overruns. However, current state-of-the-art tools do not find all errors. Manual code reviews are still required by the SDL, both to detect errors that the tools do not address and to identify opportunities for improvements in the tools. The MSDN article by Michael Howard cited in the references provides an overview of the general approach to conducting code security reviews that Microsoft teaches its engineers.

Heavy emphasis on fuzz testing is a relatively recent addition to the SDL, but results to date are very encouraging. Unlike the static code-scanning tools, fuzz testing tools must be built (or at least configured) for each file format and/or network protocol to be tested; because

of this, they are often able to find errors missed by static analysis tools.. Threat models help product teams prioritize interfaces and formats for testing. The results of fuzz testing are not totally deterministic (the tools are run for a finite number of cycles and are not guaranteed to find every bug) but experience has shown that an affordable level of fuzz testing is likely to find "interesting" bugs that might otherwise be exploited as security vulnerabilities.

5.3 Investments

Mandatory security training constitutes a significant investment for a company with an engineering population the size of Microsoft's. Training is delivered by a combination of live (instructor-led) classes and on-line material. The on-line material is especially valuable as a vehicle for delivering training to small engineering teams at sites remote from Microsoft's headquarters. The live training has proven especially effective when delivered team-wide for teams that are preparing for security pushes or other key activities – in those cases, Microsoft's experience suggests that team training results not only from the classroom training but also from conducting the security push. Security training (typically a half-day) is amplified by the fact that everyone in the workgroup is focused on security.

The central security team (SWI team) has grown significantly over the last few years as Microsoft's emphasis on security has grown. By design, the team is small relative to Microsoft's total engineering population, and emphasizes approaches that "scale" to ensure that the responsibility and resources for producing more secure software remain with product teams. Some tactics that reflect this focus on scaling include emphasis on training and tools, provision of security advisors who help the product team solve its own problems (rather than solving the problems for the team), and use of reviews (including the FSR) to provide the product team with feedback on the software's readiness for release.

5.4 Outcomes

The ultimate test of the SDL is the extent to which it removes vulnerabilities from Microsoft software. Experience – summarized in Section 4 – demonstrates that the SDL is meeting this test. Microsoft also evaluates externally reported vulnerabilities for their effect on software versions under development. Recent experience has shown that security measures planned for or implemented in new versions block attacks that are found to be effective against older versions in a growing number of cases. The recently-released Windows XP Service Pack 2 was reviewed in this way, and security changes that had been planned but not yet implemented or discussed publicly were found to eliminate a significant number of vulnerabilities reported against prior versions of Windows by security researchers external to Microsoft.

6. Conclusions

Microsoft's experience indicates that the SDL is effective at reducing the incidence of security vulnerabilities. Initial implementation of the SDL (in Windows Server 2003, SQL Server 2000 Service Pack 3, and Exchange 2000 Server Service Pack 3) resulted in significant improvements in software security, and subsequent software versions, reflecting enhancements to SDL, appear to be showing further improvements in software security.

Incremental implementation of the elements that comprise SDL has yielded incremental improvements, which we view as one sign of an effective process. 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 (see the references) have been published in an effort to share Microsoft's experience across the software industry.

7. Acknowledgements

The initial development of this paper began in late 2002 as a joint effort by the present author and Michael Howard of the SWI team. Drafts were updated as SDL evolved, and the present version was prepared over the summer of 2004. The author would like to acknowledge the contributions of Michael Howard, Matt Thomlinson, Matt Lyons, Jamil Villani, and Eric Bidstrup (all of the Microsoft Secure Windows Initiative team) to the definition and refinement of SDL. In addition to the contributors named, Scott Charney and Phil Reitinger of Microsoft and Jeannette Wing of Carnegie Mellon University provided especially helpful comments on the drafts. The author also wants to thank Martin Abadi,

Virgil Gligor, Dick Kemmerer, Chris Mitchell, Fred Schneider, Neeraj Suri, and James Whittaker for comments and suggestions on this paper

8. References

Howard, Michael, “Expert Tips for Finding Security Defects in Your Code”, MSDN Magazine, November 2003

Howard, Michael and David LeBlanc, Writing Secure Code, Second Edition, Microsoft Press, Redmond, Washington, 2003

Swiderski, Frank and Window Snyder, Threat Modeling, Microsoft Press, Redmond Washington, 2004