The Security-Inclusive Development Life Cycle

Kimberly M. Hubbard

University of Illinois Urbana-Champaign

BADM 395: IT Governance

Professor: Mike Shaw

April 29, 2007
Table of Contents

Abstract.......................................................................................................................................... 3
Purpose........................................................................................................................................... 3
Problems ........................................................................................................................................ 4
  Vulnerability ............................................................................................................................... 4
  Patch Management ...................................................................................................................... 5
  Computer Crime ......................................................................................................................... 7
Review of Process.......................................................................................................................... 9
  Microsoft’s Trustworthy Computing Security Development Process ........................................ 9
    Overview .................................................................................................................................. 10
    Education, Training and Development ................................................................................. 11
    Requirements Phase ................................................................................................................ 12
    Design Phase ........................................................................................................................... 12
    Implementation Phase .............................................................................................................. 14
    Verification Phase .................................................................................................................... 14
    Release Phase .......................................................................................................................... 15
    Support and Servicing Phase ................................................................................................. 16
  Other Security Development Processes .................................................................................... 17
    SSA ....................................................................................................................................... 17
    NIST ...................................................................................................................................... 18
Applications................................................................................................................................. 20
Results .......................................................................................................................................... 20
  Microsoft ................................................................................................................................... 20
  SSA ........................................................................................................................................... 21
  NIST ........................................................................................................................................ 22
Conclusion ................................................................................................................................... 22
Works Cited.................................................................................................................................. 23
Abstract

With the vast number of computer crimes in existence, and computer vulnerabilities on the rise, a select few computer scientists on the leading edge are taking a new approach to information security. They believe incorporating security early on, into the systems development life cycle, may be the key to making safer products that can withstand malicious attacks. This paper gives vulnerability statistics, reviews a survey of patch management costs, and analyzes the findings of a computer crime survey to outline the threat level and cost effectiveness of current security solutions. The paper then focuses on Microsoft’s Trustworthy Computing Security Development Life Cycle (SDL), and it goes into non-technical detail about activities, design methods, and techniques SDL uses to minimize vulnerabilities in their products, while pointing out the similar findings of the Social Security Administration and the National Institute of Standards and Technology.

Purpose

The purpose of this paper is to explain in non-technical detail, the latest development activities, design methods and techniques that development companies, such as Microsoft, are implementing early in the development life cycle, in order to protect the product from security vulnerabilities after release. These activities, methods and techniques from a customer perspective create a safer, better functioning product. But from an agency perspective, the ultimate goal is to create a more secure, better functioning product at less cost.
Problems

Vulnerability

With technology growing at an exponential rate, so too do the vulnerabilities in those
technologies. According to The United States Computer Emergency Readiness Team (US-
CERT), as of the first quarter of 2007, there have been nearly 24,000 internet-security
vulnerabilities reported in the US since 1988, when US-CERT was first established (National
Vulnerability Database).

Incredibly, there were more vulnerabilities reported in the year 2006 alone than in the combined
years of 1988 through 2002 [Figure 1]. And despite the government’s increase in security since
the terrorist attacks on 9-11, as of 2006 there’s been a 92% increase in the number of vulnerabilities since 2002 (National Vulnerability Database).

**Patch Management**

Currently, the typical approach to securing the vulnerabilities discovered in existing systems is to design ad hoc enhancements, otherwise known as security patches. But if not enough funds are budgeted into the development budget to cover anticipated security vulnerabilities after product release, the product is either over budget or left under secured. According to a survey of 90 organizations, the average total investment costs of patch management tools for Windows over a three-year period came to over $514,000 [Figure 2]. Open Source Software (OSS) patch management tools, although cheaper at a little over $287,000, were more expensive per system (Forbath, Kalaher, and O’Grady, 2005).

<table>
<thead>
<tr>
<th>Cost of Management Tools</th>
<th>Average Total Cost</th>
<th>Average Cost Per System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Windows</td>
<td>OSS</td>
</tr>
<tr>
<td>Patch management tools</td>
<td>$192,660</td>
<td>$107,500</td>
</tr>
<tr>
<td>Server automation tools</td>
<td>$79,400</td>
<td>$73,850</td>
</tr>
<tr>
<td>Software distribution tools</td>
<td>$242,000</td>
<td>$105,860</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$514,060</strong></td>
<td><strong>$287,210</strong></td>
</tr>
</tbody>
</table>

However, patch management tools are not the only costs incurred when implementing security patch management. Labor costs for process engineering and training, and additional costs for management oversight and configuration and inventory brought the average costs for Windows, not including patch management tools, to over $1.6 billion [Figure 3] (Forbath, Kalaher, and O’Grady, 2005).
Once again, OSS-based operating system costs were lower at less than $700,000 but more expensive on average per system (Forbath, Kalaher, and O’Grady, 2005).

Taking additional costs into account, including patching event costs, prepare and detect costs (multiplied), and total annual ongoing costs, brought annual total overall cost for Windows and OSS to over $5.7 million and $1.6 million respectively [Figure 4].

The per-system costs came to $1,622 for Windows and $2,903 for OSS-based operating systems (Forbath, Kalaher, and O’Grady, 2005).
Applying patches more frequently is a common suggestion to increase the security of a product. However, attempts to maximize product security through more frequent patch updates will increase the overall costs of patch management (Cavusoglu, Cavusoglu, and Zhang, 2006). Although necessary at some level, security patches do not have to be an organization’s only solution to securing vulnerabilities in a system or software. As will be discussed later in the paper, making changes during the initial software or system development process can decrease, and often prevent, many of the costs incurred during patchwork.

**Computer Crime**

When vulnerabilities exist, but are not detected during development, not detected in time to safely implement patches, or not detected at all until an attack occurs, even nuisance crimes can cause losses to an organization. According to the 2006 Computer Crime and Security Survey, the combined total loss of 313 respondents came to over $52.4 million during the calendar year of 2005 (Gordon, Loeb, Lucyshyn, and Richardson). The same survey found that the costliest attack to an organization came in the form of virus contamination, costing the respondents a total of over $15.6 million [Figure 5]. Coming in at second was unauthorized access to an organization’s private information, with those same respondents incurring losses of over $10.6 million. The least significant attack was exploitation of a DNS Server, at a little over $90,000 (Gordon, Loeb, Lucyshyn, and Richardson, 2006).
The survey also reported a decrease in computer crime since 1999 [Figure 6]. Although this data may lighten the mood of many a CISO, these figures were taken from a cross section of "U.S. corporations, government agencies, financial institutions, medical institutions and universities" (Gordon, Loeb, Lucyshyn, and Richardson, 2006), many of which may have legal or financial obligations for security standards. Therefore increased regulation and subsequent increased security measures may also be responsible for the decrease in crime reported.
Review of Process

The statistics just reviewed give an idea of the abundance of vulnerabilities, the threat level of computer crime, and the cost effectiveness of mainstream security solutions. But a few companies and organizations have endeavored to integrate security early into the processes by which the products are developed. Among those on the leading edge are Microsoft, the Social Security Administration, and the National Institute of Standards and Technology.

Microsoft’s Trustworthy Computing Security Development Process

There are two important goals Microsoft is pursuing with their new Trustworthy Computing Security Development Lifestyle (SDL), according to Michael Howard, who co-authored the whitepaper outlining the process. These goals are straightforward enough: “to reduce the number of security-related design and coding defects, and to reduce the severity of
any defects that are left” (Howard, 2005). The next few sections will describe, in common language, the activities, design methods, and techniques used in SDL.

Overview

The development process that Microsoft follows is rather standard for product development companies and organizations. The phases of their process include the Requirements phase, Design phase, Implementation phase, Verification phase, Release phase, and the Support and Servicing phase. But Microsoft uses them in a spiral design, allowing implementation to include revisions to the Requirements and Design phases if necessary. Where the process diverges from mainstream methods is simply where security goals are implemented and documented into the each phase of the process, ensuring the least amount of interruption to each (Lipner and Howard, 2005).

Microsoft secures their products according to four principles, Secure by Design, Secure by Default, Secure in Deployment, and Communications, collectively called the SD³+C (Lipner and Howard, 2005). The integration of the SD³+C principles into Microsoft’s standard development process is what leads to the Trustworthy Computing Security Development Lifecycle Process.

According to Microsoft’s whitepaper, these are the “brief definitions” of each (Lipner and Howard, 2005):

- 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.
- Secure by Default: in the real world, software will not achieve perfect security, so designers should assume that security flaws would 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 or accessible only to a small population of users.

- Secure in Deployment: Tools and guidance should accompany software to 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).

Education, Training and Development

Microsoft stresses that security education, security training and continued security development is key in the success of SDL. Education outside of college or university degrees may be necessary, as the type of education required is not standard in a typical college or university curriculum (Lipner and Howard, 2005). The engineers must be thoroughly knowledgeable about common security defect types, basic secure design, and security testing. It is important to know about the security features of a product, but more important to know how to build them.

Engineers’ security abilities are of such importance that Microsoft now formally requires annual security education for engineers in organizations who use the SDL (Lipner and Howard, 2005). There are references available to anyone who wishes to incorporate the training and development of security-strong engineers into their firm [Figure 7]. Microsoft has also published online resources mirroring some of the basic materials they use: [www.microsoftelearning.com/catalog/itpro.aspx#Security](http://www.microsoftelearning.com/catalog/itpro.aspx#Security). The continued training of development personnel is necessary to stay ahead of the ever-dynamic security issues encountered over time. Howard suggests engineers be retrained in the latest security findings at least once a year (2005).
Requirements Phase

In the first phase of the process, Microsoft begins their integration of security immediately, by having the production and security teams meet and assigning a “security buddy” to each product team to be the team’s security advisor all the way through to the product release. This security buddy will review plans, make security recommendations, and make sure the security team allocates resources to support the product team. The security buddy also sets goals and verifies completion of them, reporting the findings to team management (Lipner and Howard, 2005).

The production team’s responsibility in the requirements phase is to determine how security will be integrated into development as a process and also as an interface with the product. They will also determine some high-level security requirements during this phase, meeting standards and certification compliance. All of these plans will be documented and modified as needed during the development process (Lipner and Howard, 2005).

Design Phase

Security plays a major part during the design phase of the process. In general, this is where the functionality of the product is specified and in turn a design specification is created for the technical details of the functionality. The security aspects of this functionality must also be
included in the specifications, including how to implement the functionality as a secure feature. Microsoft defines ‘secure feature’ as: “ensuring that all functionality is well engineered with respect to security, […] rigorously validating all data before processing it, and several other considerations” (Howard, 2005). This security functionality is delivered in the form of a security feature, but another way to determine areas in which a security feature may be necessary is through threat modeling (Lipner and Howard, 2005).

Threat modeling is the process by which assets and the interfaces accessing those assets are located and possible threats to these assets are determined. The assessment of risk is then identified along with the countermeasures for each risk. With threat modeling, the team can recognize the most important areas for security considerations (Lipner and Howard, 2005).

Documenting the specifications, as well as the design techniques to be used in development, is also significant at this phase of the process.

Some design techniques Microsoft uses are among the following:

- Layering – structuring components to avoid circular dependencies, usually dependencies can only flow in one direction: down.
- Using strongly typed language – restricting how values having different data types can be intermixed during operations (Strong., 2007).
- Application of least privilege – ensuring that processes, users, or programs in the above defined “layers” can only access the information and resources specific to their legitimate purpose (Princ., 2007).
- Minimization of attack surface – anywhere an unauthorized user can obtain some functionality is considered an attack surface (Attack., 2007). Awareness and measurement of the attack surface can help to minimize attacks.
The Security-Inclusive Development Life Cycle

Implementation Phase

During this phase, all of the design specifications from the planning stage are implemented by coding, testing, and integrating the software. Coding is performed up to standards set by the security team, and testing will focus on the results of the threat models with the help of a few automated tools that automate tests or review code (Lipner and Howard, 2005). Some of the tools Microsoft applies are as follows:

- Fuzzing tools – a security-testing tool that uses a technique that issues structured but random and nonsensical inputs to the system, assessing the errors, if any, in the system (Fuzz., 2007).

- Static-analysis code scanning tools – detect coding flaws that result in vulnerabilities; Microsoft has developed a few of their own (Lipner and Howard, 2005).

Michael Howard, one of the whitepaper authors, makes a note to readers regarding these tools (2005):

Security tools will not make your software secure. They will help, but tools alone do not make code resilient to attack. There is simply no replacement for having a knowledgeable work force that will use the tools to enforce policy.

With that said, one of the last and most important element of the implementation phase is performing code reviews. Developers experienced in locating security vulnerabilities in source code physically review the code and correct any mistakes, removing the threat of those mistakes becoming vulnerabilities in the product (Lipner and Howard, 2005).

Verification Phase

Beta testing is performed on the fully functional product in the verification phase. During this testing, Microsoft performs what they refer to as a “security push,” a process
introduced in early 2002 (Lipner and Howard, 2005). The push involves an additional review of the code, verification of the design documentation, and making any final necessary corrections to the product for errors that have occurred during changes in the development design (Howard, 2005).

According to the white paper released on the process, these are the reasons Microsoft felt this process needed to be placed in the verification phase (Lipner and Howard, 2005):

- 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 targets 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.

**Release Phase**

Approximately six months before development is complete, the Final Security Review (FSR) is done before release of a product to the public. Under advice from the security buddy, the product team provides the security team with the deliverables necessary to complete the FSR, including a form with the following questions or tasks to be completed by the product team (Howard, 2005):

- Do you have any ActiveX® controls marked Safe for Scripting? List all the protocols you fuzzed.
- Does your component listen on unauthenticated connections?
- Does your component use UDP?
- Does your component use an ISAPI application or filter?
- Does any of your code run in System context? If yes, why?
- Do you set ACLs in setup code?
Review bugs that are deemed “won't fix” and make sure they are not mismarked security defects.

- Analyze security defects from other versions of the product and even competitors’ products to make sure you have addressed all these issues. One common question we ask is, “how have you mitigated this broad category of security issues?”
- Perform penetration testing of high-risk components, perhaps by a third-party company.

The security team then conducts the FSR, which is an independent review of the product. The objective of this exercise, according to the whitepaper, is to present 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,” or to answer, “From a security viewpoint, is this software ready to deliver to customers?” (Lipner and Howard, 2005). The final results of the review are documented and if a significant amount of vulnerabilities are found in the product, the product team must reenter the phase of development where the problem occurred, to not only correct the vulnerabilities, but to “take other pointed actions to address root causes (e.g., improve training, enhance tools)” (Lipner and Howard, 2005). Should the security team deem the product ready to be delivered to customers, a release is then planned.

Support and Servicing Phase

Since there is no such thing as a perfect product, and not all possible vulnerabilities can be anticipated before release, the support and servicing phase is necessary for addressing future discovered vulnerabilities. The product teams investigate reported vulnerabilities and release security advisories and updates when necessary (Lipner and Howard, 2005). Fortunately, these reports can then be used to reevaluate the security development process that was used. Microsoft has “dedicated staff whose job it is to perform root-cause analysis of defects,” and create what
they refer to as a “post-mortem,” to document the errors (Howard, 2005). The following
information is presented in the post-mortem (Howard, 2005):

- Name and version of file.
- Was this a design issue?
- Was this a coding issue?
- Source code diff if it's a coding bug.
- Does the bug affect other versions of the product? If yes, why? If
  not, why not? (They are both very valid questions.)
- What tests could have found this defect?
- What tools could have found this defect?
- Do we need to change our education, tools, or process to find these
  issues?

The post-mortem is used to make the necessary corrections to the SDL, in preparation for
the next development project.

**Other Security Development Processes**

**SSA**

Years before Microsoft published their whitepaper on the subject, the Social Security
Administration documented the security-inclusive development process they use in a systems
security handbook (2000). The following presents a simple, yet familiar, outline of their version
of the process (Soc., 2000):

**Systems Planning and Development**

1. Evaluating the Security Implications of New or Changed Systems Functionality
   a. Identifying Systems Changes that may Require Security Changes
   b. Analyzing the Security Implications
2. Security Teams Consult
3. User Needs Statements Prepared
   a. Access Controls
   b. Audit and Integrity Review
c. Other Controls
4. Functional Requirements and Validation Plan
5. Development and Validation of Security Features

Systems Implementation and Operation

1. As each new system is implemented, security officers in the operational components watch for potential security problems with the application, and report those problems to the SSAISSO for resolution.

2. A compilation of lifecycle products for the system is maintained and available for review by auditors, and provides complete information on the development of the application, including the security functions.

3. Periodically, each application system is subject to a recertification process, during which the application, including its security functions, is examined to ensure that it still meets SSA's objectives.

The Social Security Administration has made it policy to include this structure in their systems development process.

**NIST**

The National Institute of Standards and Technology has published a very detailed and technical guide for agencies to use in developing “cost-effective security controls” (Grance, Hash, and Stevens, 2003). Similar to Microsoft’s SDL, NIST recommends that agencies add additional security activities to their development process. The following is a very high level outline of their guide (Grance, Hash, and Stevens, 2003):

**Initiation Phase**

- Security
- Preliminary Risk Assessment
The Security-Inclusive Development Life Cycle

Acquisition / Development Phase

- Risk Assessment
- Security Functional Requirements Analysis
- Security Assurance Requirements Analysis
- Cost Considerations and Reporting
- Security Planning
- Security Control Development
- Developmental Security Test and Evaluation
- Other Planning Components

Implementation Phase

- Inspection and Acceptance
- Security Control Integration
- Security Certification
- Security Accreditation

Operations / Maintenance Phase

- Configuration Management and Control
- Continuous Monitoring

Disposition Phase

- Information Preservation
- Media Sanitization
- Hardware and Software Disposal

As evident by the outline, other than the addition of the disposition phase, NIST’s formula for security integration is relatively the same as Microsoft’s SDL. NIST, however, does not mention much about revising the process after each project to account for changes or additions in vulnerabilities.
Applications

Of the documents reviewed for this paper, only Microsoft has documented in detail the application of their SDL. Windows Server 2003, though it was not implemented with the current version of Microsoft’s SDL, was their first operating system release with some of the latest security integration processes described in the SDL (Lipner and Howard, 2005). As far as software, most of the SDL processes were implemented into Windows Server 2003. Additionally, service packs for SQL Server and Exchange Server included material as a result of security pushes made to each of the software’s then current version. And Microsoft’s last mention of a product utilizing their SDL is the Internet Information Server component of Windows Server 2003 (Lipner and Howard, 2005). The next section explains the results of the SDL inclusions of these products.

Results

Microsoft

The method of measuring the results of any new process or product must be determined by an agency before it can consider its security development process a failure or a success. Microsoft has decided to use the amount of security bulletins issued as their metric in measuring the success of the SDL. Their findings show that the number of security bulletins issued for Windows 2000 over a period of a year after release was 62, but the number for Windows 2003, which was developed using portions of the SDL, was only 24 (Lipner and Howard, 2005). The SQL Server and Exchange Server Service Pack releases had similar results. Over a period of 24 months, 16 security bulletins were issued for SQL Server 2000, compared to only 3 after the Service Pack release (utilizing SDL) for the product. And over a period of 18 months, 8 security
bulletins were issued for Exchange Server 2000, compared to only 2 after the Service Pack release (utilizing SDL) for the product (Lipner and Howard, 2005). Finally, Microsoft’s security push inclusion in the Internet Information Server component of Windows Server 2003 showed similar results. There had only been one security bulletin issued for the product in first two years of release (Lipner and Howard, 2005).

As a result of these findings, Microsoft has made the SDL mandatory for any software that is expected to be the following (Lipner and Howard, 2005):

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

**SSA**

In addition, the Social Security Administration, without any detail on which products they implemented the process, made it policy to include security in their systems development process for the following reasons (2000):

- It is more effective. Meaningful security is easier to achieve when security issues are considered as a part of a routine development process, and security safeguards are integrated into the system during its design.
- It is less expensive. To retrofit security is generally more expensive than to integrate it into an application.
- It is less obtrusive. When security safeguards are integral to a system, they are usually easier to use and less visible to the user.
NIST

Last but not least, NIST summarizes in the beginning of their recommendation document, “Including security early in the information system development life cycle (SDLC) will usually result in less expensive and more effective security than adding it to an operational system.”

Conclusion

In review of Microsoft’s findings and the motivations behind the changes the Social Security Administration and the National Institute of Standards and Technology have made to their development processes, it is fair to suggest that integrating security earlier in the systems or software development process is effective in reducing vulnerabilities after products are released. Additionally, all three entities make either direct or indirect conclusions that the security-inclusive development process is more cost effective as well. With each of their processes following similar guidelines, a coincidence should not be assumed. The logical conclusion is that the security-inclusive system or software development process is successful in creating a more secure, better functioning product less expensively.
Works Cited


