Seminar paper

“Sicherheitsmodellierung”

Security Development Lifecycle

Per Kalweit – winf9383

Winter Semester 2013/14

Lecturer: Prof. Dr. Gerd Beuster

Table of Contents

1 Introduction

The adage “Necessity is the mother of invention” sums up the development towards a more secure software environment in an increasingly tech dependent society. Because the necessity is there:

The world is more connected than it has ever been and will most likely become even more connected over time. The level of interconnectedness has created a enormous threat environment and with it an ever increasing risk for software users. The times in which mostly “script kiddies” posed a threat are long gone. Nowadays there is a large variety of entities who try to infringe software security and privacy. Cybercrime is one of the major factors in this regard. The direct cash cost of cybercrime totals at $114 billion in 2012 and time lost due to it is being valued at over $274 billion according to Norton's Cybercrime report [S1]. While figures by distributors of commercial security software have to be taken with a grain of salt, the fact that cybercrime causes severe economical damage is undeniable. The ongoing disclosures of the global surveillance by the U.S. National Security Agency (NSA) since June 2013 [S2] have furthermore shown, that intelligence agencies have to be considered as well, in regards to security and privacy in software development.

All this has consequences for software vendors. It shows how important security has become for software development in this day and age. In order to keep customer’s goodwill and preserve the company’s credibility, security vulnerabilities need to be avoided. Furthermore fixing security vulnerabilities that appear anyway Is not an option but a necessity most of the time. Since fixing (security) bugs early on in development is significantly more cost-efficient (cf. Figure 1), there is another good reason to inherit security and privacy concerns into the software engineering process as early as possible.

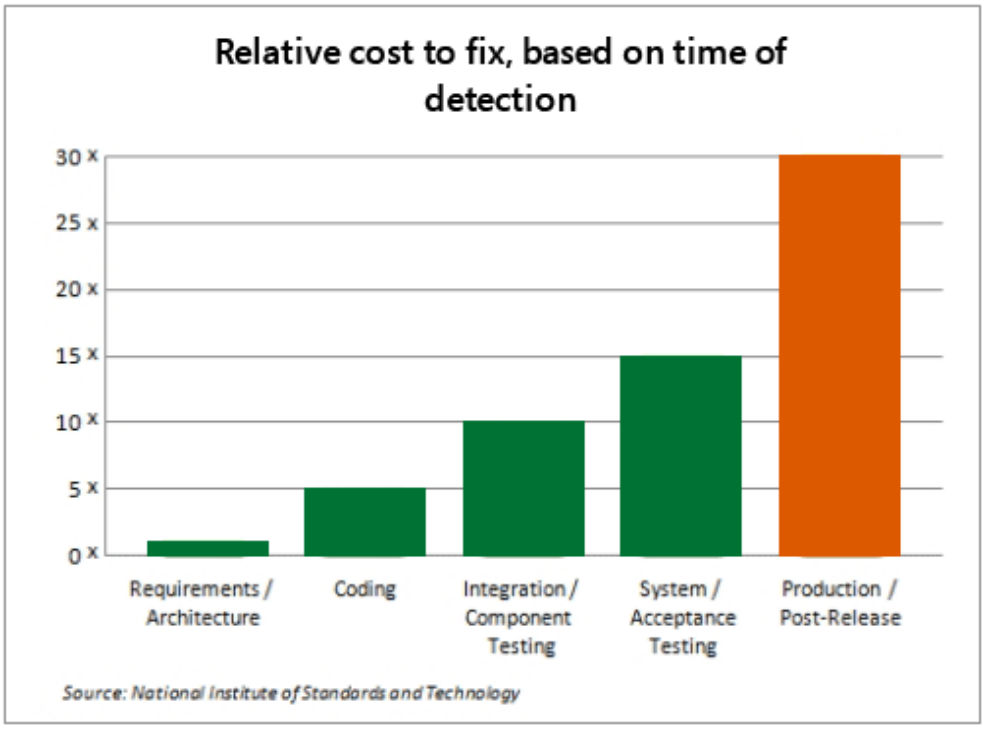

Figure 1: Relative cost to fix [software vulnerabilities], based on time of detection

1.1 Secure Software and Microsoft

Microsoft is the worlds largest independent software vendor (ISV) (according to Forbes Global 2000 [S6]) and as that producing secure software is especially important to them. Microsoft had severe (image) problems with security bugs in the past, but has stepped up and improved significantly in more recent years. This change is due to a fundamentally change in the software development process at Microsoft, incorporating security concerns into every step of the development. [S3, pp. xvii–xviii]

In order to make this approach adaptable for different projects, Microsoft formalized a process called the Security Development Lifecycle. This process is unique in the sense, that SDL isn't just theory, it is used by Microsoft for all major releases [S4] and has resulted in overall more secure software (cf. Figure 2).

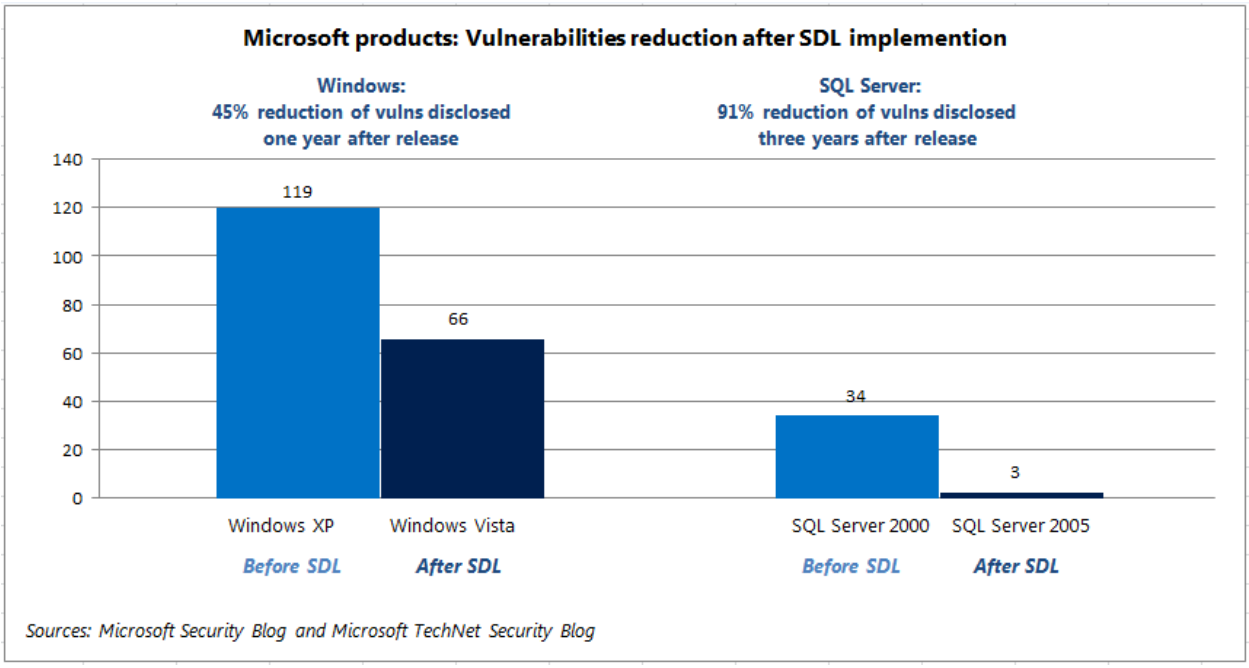

Figure 2: Microsoft products: Vulnerabilities reduction after SDL implementation

The issues of security vulnerabilities doesn't end with Microsoft software though:

According to the Secunia Vulnerability Review 2013 only 5.5% of vulnerabilities on a typical endpoint are reported in Windows operating systems. Furthermore only 14% of vulnerabilities stem from operating systems and Microsoft programs. The remaining 84% are vulnerabilities in third-party software. [S5]

This highlights the problem Microsoft has: In order to ensure the security of users under Windows operating systems, third-party software needs to become more secure. For this reason Microsoft strongly suggests, that especially large ISVs should change their software development process to something similar to SDL [S3, p. 12]. To help accomplish this goal, Microsoft provides and encourages the use of the latest implementation of SDL requirements and recommendations [S7] as well as a simplified implementation [S8].

These guidelines provide help and information to groups external to Microsoft on the application of the Microsoft SDL. The SDL process focuses on applying proven security practices at distinct points in the software development lifecycle and is based on the core concepts of education, continuous process improvement and accountability [S8, pp. 4–5].

This is aided by the principles of Secure by Design and Secure by Default. Secure by Design is the idea to consider security issues as part of the design of software

development and thus mitigating and eliminating vulnerabilities early on. Secure by Default means to increase security by giving the software the least privilege possible, disabling risky settings by default and providing backup security components in case one fails [S7, pp. 12–13].

1.2 About this Paper

This paper aims to introduce the Microsoft SDL to an audience familiar with the basics of software engineering and development. The first part covers the history of the SDL at Microsoft, it includes the different steps of how the SDL and it's predecessors evolved and highlights the change in development and mentality. The main part provides an in-depth explanation of the different steps of the SDL procedure, based on Howard and Lipner's "The Security Development Lifecycle" [S3]. Concluding there are thoughts on the applicability and future of the SDL.

2 History of the SDL

This chapter describes the the development path of Microsoft, that finally led to the Security Development Lifecycle. It offers a historic overview over the evolving landscape of threats to software, the varied approaches to combat them and brief explanations of failed and successful attempts to build more secure software.

2.1 First Steps

The first Microsoft operating systems, MS-DOS, Microsoft Windows 3.1 and Windows 95 were conceptually designed as single-user operating systems. Consequently all system resources were owned and controlled by one user, protection against other users simply wasn't necessary.

While Windows 95 was intended to be connected to corporate networks to use shared files and printer infrastructure and connect to the internet as a client, the primary focus concerning security was put on the browser. But due to a much smaller environment for threats, the understanding of security and privacy needs was vastly different from what it is today [S3, p. 27].

Excursion: Trusted Computer System Evaluation Criteria

The United States government's National Security Agency (NSA) began the development of a set evaluation criteria with the intention of characterizing security features and assurance of operating system software in the early 1980s. The resulting criteria is called the Trusted Computer System Evaluation Criteria (TCSEC) also frequently referred to as the Orange Book after the color of its cover [S3, p. 28]. TCSEC categorizes the security of computer systems in a hierarchic system with four divisions, from D to A, which are broken into subdivisions called classes [S9].

Most commercial operating systems at the time achieved "Class C2" evaluations [S3, p. 28]. Security Evaluations of other countries followed (ITSEC in Europe) and were eventually succeeded by the Common Criteria for Information Technology Security Evaluation in the late 1990 (internationally recognized as ISO Standard 15408) [S10].

Returning to the history of the SDL leads to the first product for which security was a significant part of the design: Windows NT 3.1. The core design team Windows NT 3.1 came from Digital Equipment, a lot of them experienced with work on a system targeted at the higher "Class B2" TCSEC evaluation. As one of the relatively secure multiuser system of its time it saw a lot of use as a print, file or internet server (HTTP, FTP, etc.). It was also widely used as a desktop client, as it was considered to be more robust than Windows 95 [S3, pp. 27–29].

2.2 New Threats

With the explosive growth of the internet in the mid-1990s started the evolution of an industry specialized in discovering security vulnerabilities. Discovered vulnerabilities of Netscape and Internet Explorer [S12] received wide publicity.

While a lot discoveries were made by security companies and used to build up credibility, hostile attacks were there since the very beginning of the commercial internet [S11].

Until 1998, Microsoft had taken an ad hoc approach to dealing with discovered software vulnerabilities. Individual product teams had to communicate with vulnerability finders and release fixes by themselves.

This changed partly with the creation of the security response team in mid 1998. It's main responsibility was a centralized well-known e-mail address for security researchers and a single website to communicate security updates.

In addition to the security response team an internal Security Task Force was formed, to find underlying problems and develop recommendations to avoid and reduce vulnerabilities over time. These recommendations constitute the earliest predecessor of the SDL. [S3, pp. 29–30]

2.3 Windows 2000 and the Secure Windows Initiative

When the security task force released a report, simultaneous to the bug fixing phase of Windows 2000, it was clear, that the increasing number of vulnerability reports had to be dealt with. On the basis of recommendations of the security task force, key steps were made:

Microsoft developed their first automated static analysis tool with PREfix that could detect some classes of security vulnerabilities (e.g. some buffer overruns). In addition to that they formed a dedicated penetration test team, to find more vulnerabilities in the base code.

But the most significant step was the creation of a dedicated security program management team: The secure Windows Initiative (SWI). They worked together with product teams by reviewing component design and code, making recommendations on how to improve the components security.

Lastly it was decided, that security vulnerabilities would be treated as "ship-stoppers". This shows the increasing commitment of the Windows division management towards improving security in Microsoft software. It sent a clear message to all contributors, that the focus of development was shifting. [S3, pp. 30–32]

2.4 Windows XP

After the release of Windows 2000 and the work on its successor Windows XP it became increasingly obvious, that the SWI, staffed with under five members, was not sufficient to review the existing design and code. As a result the team made a fundamental change:

They would still be available for specific design or coding issues concerning security. But instead of reviewing code or design themselves, they would focus on teaching the teams how to look for security vulnerabilities. The goal was to change toward a more scalable approach.

To get the teams used to considering security in their projects, the SWI organized team-wide security days, so called “bug bashes”. Such a day would start with security training, followed by code reviews or security testing. In order to further increase acceptance, the SWI handed out prizes for the “best security bug” filed that day. [S3, pp. 32–33]

2.5 Security Pushes and Final Security Reviews

The second half of 2001 was a difficult time for Microsoft concerning security. There was the Code Red Internet worm, exploiting a vulnerability in the Internet Information Services (IIS) 5 Web server component under Windows 2000 systems [S13]. Later in September, the Nimda worm exploited another vulnerability in IIS [S14].

Seeing as these were not the only discovered exploits, Microsoft’s top management worked on plans to fundamentally change the ways to address security and privacy. This led to the Trustworthy Computing (TwC) initiative, aimed to mobilize Microsoft’s staff and improve the overall software quality. An e-mail by Bill Gates to all of Microsoft employees launched the TwC in 2002.

Meanwhile the SWI was working on a new method to improve software security at launch: The security push. This method was first applied on the .NET Framework common language runtime. After delaying the release of the product, the SWI initiated a concentrated effort of code reviews and security testing until the rate of filed security bugs diminished so far, that further searching would have been unproductive.

The security push was so effective, that a similar method was used before the launch of the considerably more extensive Windows Server 2003. After the push was finished, the question, whether or not the push was worthwhile emerged. This was answered with a process, which was later called the Final Security Review. The SWI reviewed filed security bugs in the context of vulnerabilities affecting prior Windows versions and competitor products as well as penetration tests by outside contractors.

The outcome was positive and the Final Security Review a new tool for large software. [S3, pp. 33–36]

2.6 Formalizing the SDL

While the security pushes and final security reviews posed good methods, the process that the SWI team followed was still ad hoc. The practices were guided by an “oral tradition” and it wasn’t entirely clear what the process itself was.

In order to clear up this confusion the process had to be formalized. The SWI team held meetings with Microsoft’s senior managers, in early 2004. Together they reviewed the achievements since the first pushes and compiled the requirements to apply the ad hoc process of training, security pushes and final security reviews consistently and effectively.

The result was a formally defined Security Development Lifecycle in its first appearance. From there on essentially every Microsoft product had to meet the requirements of the SDL. By July 2004 over half of Microsoft’s engineers had completed the security training mandated by the SDL and the official requirements for SDL compliance were available on an internal web site.

Since then the SDL was updated regularly by the SWI team and has reached the current version of 5.2 in May 2012. [S3, p. 36–37][S7]

3 The SDL Process

This chapter gives an in-depth description of the Microsoft Security Development Lifecycle (SDL), it is the core of this paper. Each subpart concerns one stage of the SDL and it's requirements as laid out in Michael Howard and Steve Lipner's "The Security Development Lifecycle" [S3].

3.1 Education and Awareness

Security education of the engineering workforce is one of the key success factors of the SDL at Microsoft. If an engineer doesn't know the common security bug types or basics of secure design and security testing, there is a high chance that he will not produce secure software.

At Microsoft it is understood, that understanding security features, such as how encryption algorithms work, while useful, aren't sufficient to produce secure software. The problem lies in the fact, that most schools and universities teach security features, but not how to build secure software. Since there is no guarantee, that experienced engineers might have picked up the skills to build secure software, it isn't possible to assume that any hired worker understands how to build security defenses into software. The conclusion is simple: Software engineers need to be further educated in making secure software.

To properly educate the engineering workforce, the education has to be ongoing. Threats, research and mitigations change at rapid rate, consequently training has to be updated accordingly. Workers should attend security training at least once a year. Since it isn't useful to teach the basics of secure software development more than once, a security curriculum to address the more specific needs of different disciplines needs to be defined. As an example, these are some classes teached at Microsoft:

- **Performing Security Code Reviews** Few people know how to properly review code for security bugs. This class teaches some critical skills, like being suspicious of incoming data and ranking system entry points by their "attackability".

- **Exploit Development** This class teaches how to create exploit code to take advantage of vulnerabilities. Obviously not to attack real systems, but to educate on how dangerous security bugs can be and to help understanding security vulnerabilities better.

- **Customer Privacy** This class focuses on protecting customer data. Including legal aspects of privacy, the data lifecycle, global privacy policy, data integrity and enforcement.

There are different types of training. At Microsoft, live training sessions have proven to be very effective, but they are problematic if a large number of employees need to be trained. In this case, the recording of training sessions can help to increase the reach of a class.

When making a new class, the objective and target audience need to be clear. After an security expert built the class other experts need to review the material for accuracy and applicability. Feedback is important in order to fine-tune timing and content. It's application can be incorporated into the iteration process to keep the class up to date. [S3, pp. 55–66] [S17, pp. 26–31]

3.2 Project Inception

In Microsofts experience, a good project start leads to a smoother final security review and an overall more secure product. For this reason the project inception holds high importance.

The project inception is split into two discrete steps:

3.2.1 Determine Whether the Application Is Covered by SDL

First needs to be determined whether SDL should be applied to the product at hand. Ultimately, all software will benefit from following the SDL, but there are some criteria which make following the SDL mandatory:

- Any product that is commonly used or deployed within a business (e.g., e-mail and database servers).
- Any product that regularly stores, processes, or communicates personally identifiable information (PII) (e.g., financial or medical customer information).
- Any products or services that target or are attractive to children (because of various child online protection laws, such as the Children's Online Privacy Protection Act (COPPA 1998)) [S15]
- Any product that regularly touches or listens on the Internet:
    - Always online: services provided by a product that involves a presence on the Internet (e.g., instant messaging software)
    - Designed to be online: browser or mail applications that expose Internet functionality (e.g., Web browsers, e-mail clients)
    - Exposure online: components that are routinely accessible through other products that interact with the Internet (e.g., games with multiplayer online support)

- Any product that automatically downloads updates.

3.2.2 Assign the Security Advisor

The development team needs a security person to guide it through the SDL process with the aim of successfully completing the final security review.

If there is a central security team, or an engineering quality team, it is advised to nominate someone from that team to be the security advisor. The person nominated to be the security advisor should generally have deep security as well as project management skills. While the former is important, experience at Microsoft has shown, that it is advisable to lay the focus on the latter.

The tasks of the security advisor include: [S3. pp. 67–74]

- Acting as a point of contact between the development team and the security team.
- Holding an SDL kick-off meeting for the development team.
- Holding design and threat-model reviews with the development team.
- Analyzing and triaging security-related and privacy-related bugs.
- Acting as a security sounding board for the development team.
- Preparing the development team for the FSR.
- Working with the reactive security team.

3.3 Define Design Best Practices

The design phase hold special importance in software development as a lot of problems can be avoided by design. To accomplish this regarding security in software, the SDL proposes two methods:

3.3.1 Common Secure-Design Principles

There are numerous secure-design principles, the SDL recommends those created by Saltzer and Schroeder in their paper, “The Protection of Information in Computer Systems”: [S16]

- **Economy of mechanism** Keep the code and design simple. The less complex the software, the smaller the likelihood of bugs in the code.
- **Fail-safe defaults** The default action for any request should be to deny the action. Thus, if the user request fails, the system remains secure.

- **Complete mediation** Every access to every protected object should be validated.
- **Open design** Open design, as opposed to “security through obscurity,” suggests that designs should not be secret.
- **Separation of privilege** Do not permit an operation based on one condition. Examples include two-factor authentication, and, at a higher level, separation of duties.
- **Least privilege** Operate with the lowest level of privilege necessary to perform the required tasks.
- **Least common mechanism** Minimize shared resources such as files and variables. Code that uses only local variables is more robust and maintainable than code that uses global variables.

3.3.2 Attack Surface Reduction

Every application has accessible code, because this code is potentially vulnerable to users with malicious intent. “The attack surface of a software product is the union of code, interfaces, services, and protocols available to all users, especially what is accessible by unauthenticated or remote users.” [S3, p. 78] The SDL process suggests to make the reduction of this Attack surface an integral part of designing software. The process to achieve this goal is shown in Figure 3. [S3, pp. 75–91]

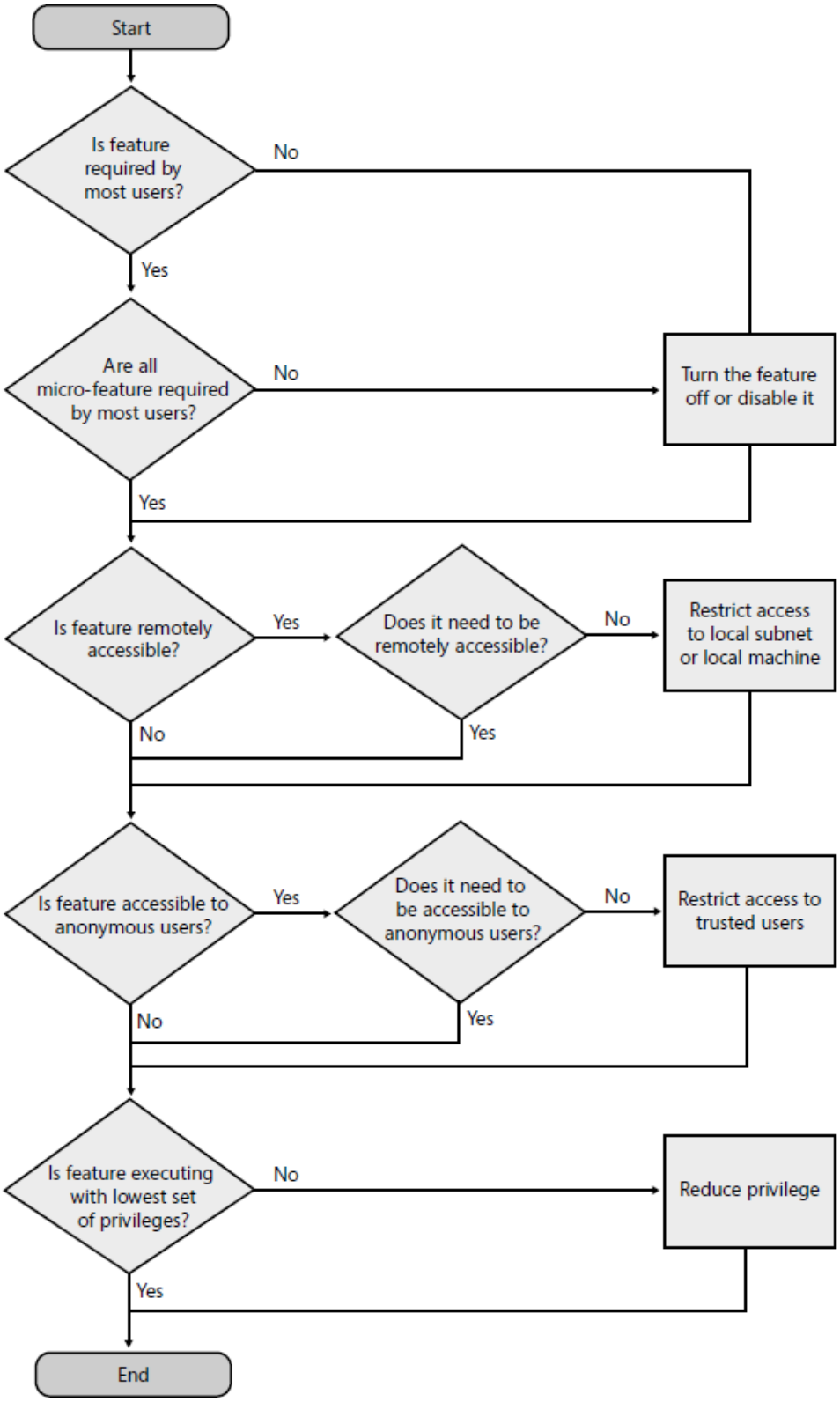

Figure 3: Steps to reduce the attack surface

3.4 Risk Analysis

Risk Analysis as proposed by the SDL process means building a threat model. A threat model is there to understand the potential security threats to the system, determine risk, and establish appropriate mitigations. It can be used to find design issues early on, which in turn reduces development cost. This is the case, because defects in early parts of development make changes in all subsequent parts necessary.

The threat-modeling process is split into a number of steps:

1. Define use scenarios.
2. Gather a list of external dependencies.
3. Define security assumptions.
4. Create external security notes.
5. Create one or more DFDs of the application being modeled.
6. Determine threat types.
7. Identify the threats to the system.
8. Determine risk.
9. Plan mitigations.

The threat model can be used to aid code review. Since the threat model delivers a list of entry point to the system, code reviewers can focus their attention on code that is remotely and anonymously accessible.

The threat model can also aid testing in a similar manner: Since planned mitigations are known, penetration tests can be built around attacking these specific mitigation techniques. [S3, pp. 101–132] [S18, pp. 25–37]

3.5 Secure Coding Policies

The SDL mandates specific coding practices. The following coding best practices must be adhered to for all code: [S3, pp. 143–151] [S17, pp. 127–170]

- **Use the latest compiler and supporting tool versions.**
- **Use defenses added by the compiler.** The /GS flag (Buffer Security Check) is an example of defense added by the compiler:

    - Buffer overrun check by placing a random value on the stack before the return address if the value changed after the function has returned, the application aborts.
    - The stackframe is rearranged by the compiler, so that potentially exploitable variables such as function pointers won't overrun other constructs.

other suggested options are /SAFESEH for the linker, which checks if a called exception handler is valid (not overwritten) and /NXCOMPAT which lets the linker check, if the file was tested to be compatible with the Data Execution Protection (DEP) feature in Microsoft Windows.

- **Use source-code analysis tools.** Source-code analysis tools on their own don't make software more secure, but they can help to scale the code review process and enforce secure-coding policies.
- **Do not use banned functions.** SDL provides a list of banned functions and suggested substitutes, for example, strcpy is to be replaced by strcpy_s, strncpy by strncpy_s. [S3, pp. 241–249]
- **Reduce potentially exploitable coding or design constructs.** SDL proposes to think about potentially exploitable coding constructs. For example, creating an object under Windows with an empty access control list (acl), which means the object has no protection.
- **Use a secure coding checklist.** Although a simple checklist can't guarantee secure code by itself, it can provide a minimum-security bar just by following it.

3.6 Secure Testing Policies

Testing is an integral part of software development. It is important to validate written code and to minimize the amount of vulnerabilities. Nonetheless it is important to keep in mind, that it isn't possible to test security into products written in an insecure manner.

The SDL testing phase consists of the following steps: [S3, pp. 153–167][S17, pp. 567–613]

1. **Fuzz testing** Originally a method to find reliability bugs, it can be used to make out certain classes of security bugs as well. Fuzz testing means testing the reaction of an application to input of malformed data. Every crash or unexpected error needs to be investigated and interpreted with security in mind. If a large number bugs occur while fuzz testing, it is recommended to start a deep code review instead.

2. **Penetration testing** Penetration testing (pentesting) is designed to find vulnerabilities in information systems. SDL does not involve a description of pentesting and advises to consider using a third-party company to perform the pentest.

3. **Run-time verification** Run-time verification is testing while the application is running, using tools such as AppVerif to find bugs like Heap-based memory leaks or uninitialized variables.

4. **Re-reviewing threat models** Threat model should be review after testing, to make sure they are still accurate and cover all functionality of the software.

5. **Reevaluating the attack surface** Reevaluating the attack surface while testing allows to focus further testing and code reviews on high-risk areas. Corrective actions can be taken, like disabling a component by default. All changes to the attack surface should be properly documented.

3.7 The Security Push

The security push was one of the major milestones in the development of the SDL (cf. Chapter 2.5 Security Pushes and Final Security Reviews) but the concept behind them is flawed: SDL aims to reduce the number and severity of vulnerabilities in the first place, but security pushes work more like an afterthought, hunting for bugs late in the process.

The SDL has a place for the security push, which is a push with a primary focus on legacy code (code created before the current development cycle). However, the security push is not to be viewed as a quick fix for insecure code.

A security push should be planned in the beta timeframe, with a final beta test after the security push is over. The security push is not restricted to code. The main tasks during the push are as follows: [S3, pp. 169–179]

- **Training** The software development team needs training on how to approach the security push. Additional technical-security training might be worthwhile for some employees.

- **Code reviews** Reviewing legacy code is unpopular among developers, unfortunately code can be vulnerable regardless of its age. To review code, each source code file is assigned to an owner, and a review priority, based on the threat model. The owner swap their code with another owner, so that no owner reviews her own code.

- **Threat-model updates** The threat models need to be reviewed once again, to make sure that they are complete and correct. Examples on what can be checked are:
    - Check if the data flow diagram incorporates all design changes.
    - Make sure the list of entry points into the system is complete.
    - Make sure all the sensitive data stores are protected correctly, this often includes an access control list review.
- **Security testing** Testing during the security push differs from the previously introduced testing step in the sense that only highest-risk components are focussed. A final validation, to make sure fuzz tests are in place for all parsed file formats, is also possible.
- **Attack-Surface scrub** The attack surface gets reevaluated again, to make sure the software has "good security hygiene". Examples for attack-surface scrub criteria are:
    - Count all the open ports, sockets, SOAP interfaces, remote procedure call (RPC) endpoints, and pipes. Are they all needed by default?
    - Verify that all unauthenticated network entry points are needed. Can they be authenticated by default?
    - Verify that all network entry points are restricted to the correct subnet or set of trusted source addresses.

3.8 Final Security Review

As explained in 2.5 "Security Pushes and Final Security Reviews", the final security review (FSR) seeks an answer to the question whether or not the product is ready to be released from a security and privacy standpoint. An FSR is a way to determine overall ship quality and can take a long time. A failed FSR must be evaluated to determine the severity of the issues and senior management must decide on how to resolve it, if the product team can't.

The FSR process consists of the following components: [S3, pp. 181–185]

- **Product team coordination** This step is not technical, the goal is to fill out a questionnaire. Examples of questions are:
    - Is this product standalone or a service/management/feature/add-on pack?

    - Is any part of the product network-facing?
    - When was the security push, and how long did it take?
    - Where is the attack surface documented?

- **Threat models review** The threat models need to be reviewed again to make sure they are accurate, up to date, and have appropriate mitigations in place. The date of the last change can be an indicator for the model relevance, an old model will probably be out of date. An inaccurate or incomplete data flow diagram is also an indicator for a inaccurate model.
- **Unfixed security bugs review** All filed bugs are reviewed to account for human error. Examples are security bugs marked as “Won’t Fix” that are above the specified priority threshold for bugs that need to be fixed before release. After all bugs have been reviewed, the results have to be analyzed to determine which bugs have to be fixed.
- **Tools-use validation** SDL requires the use of certain tools and compiler options [S3, pp. 259–268]. This step validated compliance with appropriate tools and compiler options.

4 Conclusion

The need for more secure software is still there and the Microsoft Security Development Lifecycle proposes a solid process to integrate security and privacy concerns into software development.

The SDL paid of for Microsoft, not only did it reduce the number and severity of vulnerabilities in their software, it has reduced costs by fixing security issues before the code was deployed and meant an overall increase to Microsoft's image.

On the other hand the SDL process adds a lot complexity to the software development and means a large investment for other developers. With numerous tests, reviews and extensive educational ambitions a lot of additional pressure is put on developers.

It’s likely that many companies are unable or unwilling to adopt the full SDL. But since addressing security becomes more of a necessity these days, the simplified implementation of the SDL might be the compromise low- and mid-range developers are or will be looking for.

6 Sources

[S1]: Norton Cybercrime report 2012
http://us.norton.com/cybercrimereport
as at April 6th, 2014

[S2]: The Guardian: The NSA Files
http://www.theguardian.com/world/the-nsa-files
as at April 15th, 2014

[S3]: The Security Development Lifecycle
by Michael Howard, Steve Lipner
Microsoft Press, Redmond, 2006
ISBN 10: 0-7356-2214-0

[S4]: Security Development Lifecycle, Official Website
https://www.microsoft.com/security/sdl
as at April 6th, 2014

[S5]: Secunia Vulnerability Review 2013
http://secunia.com/?
action=fetch&filename=Secunia_Vulnerability_Review_2013.pdf
as at April 6th, 2014

[S6]: Forbes Global 2000, 2013
http://www.forbes.com/global2000/#page:1_sort:0_direction:asc_search:_filter:Software%20%26%20Programming_filter:All%20countries_filter:All%20states
as at April 6th, 2014

[S7]: Microsoft SDL Process Guidance Version 5.2
http://www.microsoft.com/en-us/download/details.aspx?id=29884
as at April 6th, 2014

[S8]: Simplified Implementation of the Microsoft SDL
http://go.microsoft.com/?linkid=9708425
as at April 6th, 2014

[S9]: Department of Defense: Trusted Computer System Evaluation Criteria (TCSEC)
http://csrc.nist.gov/publications/history/dod85.pdf
August 15th, 1983

[S10]: Common Criteria Project, Official Website
http://www.commoncriteriaportal.org/
as at April 6th, 2014

[S11]: CERT Advisory: wuarchive ftpd Trojan Horse
http://www.cert.org/historical/advisories/CA-1994-07.cfm
as at April 6th, 2014

[S12]: CERT Advisory: JavaScript Vulnerability
http://www.cert.org/historical/advisories/CA-1997-20.cfm
as at April 6th, 2014

[S13]: CERT Advisory: "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL
http://www.cert.org/historical/advisories/CA-2001-19.cfm
as at April 6th, 2014

[S14]: CERT Advisory: Nimda Worm
https://www.cert.org/historical/advisories/CA-2001-26.cfm
as at April 6th, 2014

[S15]: COPPA - Children's Online Privacy Protection Act
http://www.coppa.org/coppa.htm
as at April 6th, 2014

[S16]: The Protection of Information in Computer Systems
Jerome H. Saltzer, Michael D. Schroeder
revised April 17th, 1975
Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology Cambridge, Mass.
http://web.mit.edu/Saltzer/www/publications/protection/

[S17]: Writing Secure Code, Second Edition
Micheal Howard and David LeBlanc
Microsoft Press, Redmond, Washington, 2003
ISBN 10: 0-7356-1722-8

[S18]: Threat Modeling
Frank Swiderski, Window Snyder
Microsoft Press, Redmond, Washington, 2004
ISBN 10: 0-7356-1991-3

7 Figures