Software Security Checklist for the Software Life Cycle

David P. Gilliam, Thomas L. Wolfe, Josef S. Sherif
Jet Propulsion Laboratory, California Institute of Technology
david.p.gilliam@jpl.nasa.gov, thomas.l.wolfe@jpl.nasa.gov, josef.s.sherif@jpl.nasa.gov

Matt Bishop
University of California at Davis
bishop@cs.ucdavis.edu

Abstract

A formal approach to security in the software life cycle is essential to protect corporate resources. However, little thought has been given to this aspect of software development. Traditionally, software security has been treated as an afterthought leading to a cycle of 'penetrate and patch.' Due to its criticality, security should be integrated as a formal approach in the software life cycle. Both a software security checklist and assessment tools should be incorporated into this life cycle process. The current research at JPL addresses both of these areas through the development of a Software Security Assessment Instrument (SSAI). This paper focuses on the development of a Software Security Checklist (SSC) for the life cycle. It includes the critical areas of requirements gathering and specification, design and code issues, and maintenance and decommissioning of software and systems.

1. Introduction

A recent article, "Why Is Software So Bad" [1] points out that bad habits and inadequate software life cycle processes have led to the development of poor software. In the last 10 years of advancements in software engineering and the development of tools, progress in improving the quality of software is still lagging. In particular this assessment can be made with respect to security. That is, "Why is software so insecure and vulnerable?"

Gary McGraw points out the essential dilemma for security, "There is no such thing as 100 percent security. In fact, there is a fundamental tension inherent in today's technology between functionality (an essential property of any working system) and security (also an essential in many cases)."[2] This point is still true today. Why? And why is integrating security into the software life cycle so critical, but still lagging behind other disciplines? Integrating security into the software life cycle process until now has been a haphazard process. It requires trained experts and dedicated resources.

A life cycle process that includes security assurance is needed for improving the overall security of software. Implementing this process is the goal of this research effort. The research has 5 foci that are integrated into a Software Security Assessment Instrument (SSAI). The SSAI contains: 1) a Software Security Checklist (SSC), 2) a vulnerability matrix that categorizes vulnerabilities and exposures, 3) a Flexible Modeling Framework (FMF) for verification of requirements, 4) a Property-Based Tester (PBT) for testing for vulnerabilities, and 5) a collection of Security Assessment Tools (SATs) with a description and explanation of their use for assessing the security of software. While this research has been reported previously, the current work that is new is the development of the SSC.[3,4,11,12] The SSC is an instrument to guide organizations and System Engineers (SE's) in integrating security into the software life cycle.[4] The process from requirements gathering to system test and integration, maintenance and even decommissioning is covered by this SSC. The SSC has two phases. Phase one is a security checklist for the software life cycle as described above. Phase 2 is a security checklist for the external release of software.

2. Current State of Software Security

There are several reasons for the current state of software development. Two key issues are, 1) "The attitude today is that you can write any sloppy piece of code and the compiler will run diagnostics;" 2) "The constant demand for novelty means that software is always in the bleeding-edge phase, when products are inherently less reliable." To these reasons should also

Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)
1080-1383/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

be added lack of skilled developers and training, and lack of resources such as code analyzers that can check the security of software and systems. As McGraw points out, "Security is like fault tolerance, a system-wide emergent property that requires much advance planning and careful design."[5,6]

2.1 Current research

Organizations and companies are now recognizing the importance of security in the life cycle from network security, to system security and application security as an integrated end-to-end process.[5,7,9] Microsoft, among others, has instituted a security initiative that is corporate-wide. It is hoped that these efforts will produce software that is inherently more secure.[14]

The highly volatile computing environment requires that security be viewed as a continuing process to meet the changing needs of the environment. Even with good requirements, security design flaws are still prevalent.

Several recent studies have shown that the risk of not integrating security into the software life cycle can have highly negative impacts on a company. Both Cigital and @Stake have done studies that show that integrating security in the software life cycle has proven benefits both in cost and image.[6,8,9]

Currently, companies like Citigal and @Stake are offering assistance and tools for verifying the security of software.[5,6,8] They include security fault injection tools and attack trees. Carnegie Mellon University has been working on the modeling of potential attacks against software.[10]

These tools are a start in the fight to mitigate security risks but more needs to be done. To date, an end-to-end life cycle security risk mitigation instrument is not known to be available commercially.[1] The current work in progress on the SSAI addresses this need by creating a security risk mitigation instrument that addresses these weaknesses in the software life cycle. Included is an SSC for the life cycle as described below, a vulnerability Matrix (Vmatrix), a flexible modeling framework (FMF) with model-based verification (MBV,) and a property-based testing (PBT) tool for the requirements specification, development and testing phases of the life cycle (previously reported).[3,4,11,12,15] The FMF uses Model Checking and SPIN model checker to check for properties in the requirements specifications that lead to vulnerabilities or unwanted exposures. The PBT tool can then verify the code that these properties have not been re-introduced into it.

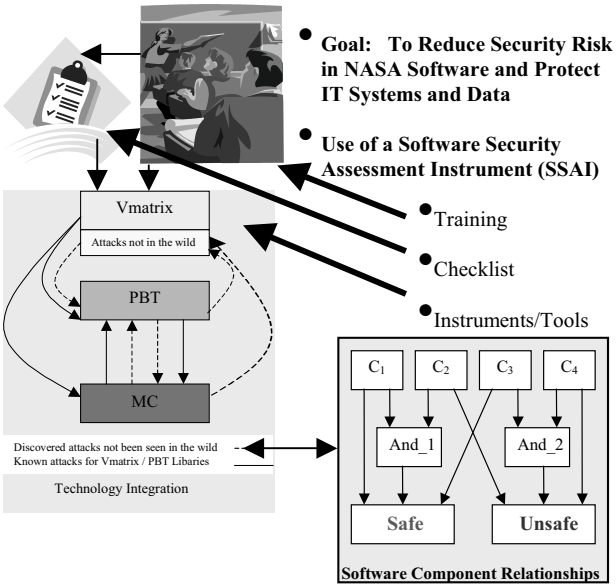

Figure 1: Software Security Goal and Instrument

2.2 Exploits and exposures

The exploits used to break into systems vary. However, they can be grouped into broad categories of similarity. The current research effort is developing an SSC as part of an SSAI contains an appendix describing a list of common vulnerabilities and exposures. A brief list includes the following:

1. *Environment variables*: Variables that encapsulate information that does not change across executions of a program. On UNIX systems, the PATH environment variable lists the directories to be searched for a named executable. Regardless of how many different executables are searched for, the PATH variable's value does not change.

2. *Buffer Overflows*: Overflowing a memory stack so that the program will execute the data after the last address in the stack, usually an executable program that establishes a root or command line shell giving the attacker full control of the system. Others are heap overflows that contain code that the program can branch to via function pointers, and data overflows to alter variable values in conjunction with executing code contained in environment variables.

3. *Data as Instructions* or *Script Injections*: Using scripting languages to include information with executable code which the system executes due to improper input checking.

IEEE COMPUTER SOCIETY

4. *Numeric Overflows*: Giving a larger or smaller value than expected. This assumes that a particular values stays within established bounds. The concept is to look for numbers that can be more than 2^32 or greater, or the maximum integer

5. *Race Conditions*: Sending a string of data before another is executed. The most common type is the "Time of Check to Time of Use" flaw. Another is masquerading or "Man-In-The-Middle" attacks.

6. *Network Exposures*: Assuming that clients will check messages sent to a server adequately. Remote commands and executables provide the majority of examples of this type of exploit ("r" protocols like rsh, rlogin, and especially rexd).

7. *Information Exposure*: Exposing sensitive information to unauthorized users that can be used to compromise data or systems. For example: 1) non-secure transmission of sensitive information such as human resource data that can be used for social engineering; 2) Use of clear text user ID's and passwords; and 3) weak encryption schemes for access.

8. *Operational Misuse*: Operating a system in a non-secure mode. Using standard accounts with blank passwords, or providing open shares giving everyone access. Anonymous file transfer is common where users are given read/write access to a set of directories or files.

9. *Default Settings*: Default software settings may present a risk if they require user intervention to secure them. For example, Root or Administrator accounts that do not require an initial strong password also present risks if they are not set when installed such as Windows NT and 2000. Also, applications using open ports that neither the system nor application check for authentication, present potential risks. Known examples are: SunOS's use of "+" in the default /etc/hosts.equiv file; or leaving the uudecode alias in the mail alias file.

10. *Programmer Backdoors*: Unauthorized access paths left by developers of the software for easy access. If web services are included, this list greatly changes and expands as shown by Jaquith.[8] The evaluation of security flaws in 45 commercial applications, found security design flaws in 70 percent of the defects observed, with nearly half of these classified as serious.

The appendix of common vulnerabilities can be used in concert with the SSC to mitigate these exposures and vulnerabilities. Its use with other instruments and tools in the SSAI can lead to the development of software that is less prone to these types of defects.

2.3 Software security assurance

In view of these issues, a life cycle process that includes security is essential. Security needs to be part of an end-to-end life cycle process, commonly called "cradle-to-grave." "Fixing applications post-attack is expensive, both financially and in terms of . . . reputation,"[15] while integrating security into the life cycle is a true revenue loss preventative.

It is highly cost effective for an organization to document their security policies, requirements, guidelines and procedures. Additionally, education and training of SE's, developers, System Administrators (SA's), etc., are essential. There should be a corporate-wide proactive and viable enforcement policy for assuring security in the life cycle. Security policies, standards, guidelines and best practices should be inherent in the process. What is critical is that security be integrated into the life cycle process. Critical systems such as life support and nuclear control systems especially need this type of process.

3. Guidelines for Generating an (SSC)

The question is, then, how to integrate security into the life cycle? Knowing that defects can be reduced and public image enhanced by providing products that are more secure than competing products can drive the efforts towards development of more secure software.

The first step is to perform a security risk analysis and then identify security issues and requirements. The second step is to use an SSC instrument for all phases of the life cycle. The risk analysis and requirements should then drive the rest of the life cycle with traceable and verifiable security requirements throughout.

To what extent is security required and what is the level of risk versus cost that is willing to be accepted? Second the software should have a risk level rating. This rating will provide those using it an awareness of the overall security of the software. This is a critical issue when integrating software with other software and system components, especially in mission and safety critical systems. A security risk rating is a particularly useful item for re-usable code. The current SSC under development has two phases. One phase addresses the software development and maintenance life cycles. The second phase addresses the external release of software.

What are some of the issues that need to be included in a SSC? Table 1 below describes some critical areas that can be used for generating an SSC. The list in Table 1 is provided as one example for generating an SSC. The list can be extended, modified, or even replaced. What is critical is that there be either a formal or informal process for verification of security requirements.

IEEE COMPUTER SOCIETY

Table 1: Items for Potential Consideration and Inclusion in a Software Security Checklist (SSC)

1	Introduce a walkthrough, security audit review or a formal security review in every phase of the software life cycle development.
2	Establish security metrics during the software life cycle and a trace matrix for security requirements.
3	Determine stakeholders, and elicit and specify associated security requirements for each stakeholder
4	Determine context and potential usage of software product along with the operating environment and specify requisite security requirements.
5	Make available to programmers, developers, reviewers and test teams the vulnerabilities and potential exposures associated with programming languages and operating systems before the architectural design phase.
6	Set up security parameters for access to services such as *ftp* service where anonymous ftp is allowed but with write only and no read or list to the incoming directory and read only for outgoing directory
7	Check for sources of software security risks such as inconsistencies in requirements and in design, reusable programs and other shrink-wrap software. Use of requirements tools, modeling tools, etc. can aid in this area.
8	Avoid the use of unsafe routines such as sprintf(), strcpy/cat(), *gets* and *fgets* in coding.
9	Check the security of any middleware in the program.
10	Check for architectural-specific vulnerabilities and how data flows through the code.
11	Check for implementation-specific vulnerabilities such as Race Conditions, randomness problems and buffer overflows.
12	DO NOT allow programmer backdoors or unauthorized access paths that bypass security mechanisms.
13	Avoid storing secrets like passwords in the code or use weak encryption schemes
14	Identify all points in the source code where the program takes input from users.
15	Identify all points in the source code where the program takes input from another program or un-trusted source.
16	Investigate all sources from which input can enter the program such as GUI, network reads, etc.
17	Check API (Application Program Interfaces) calls to security modules or interfaces.
18	Investigate secure connections. Verify that they actually are secure and connect as indicated to the systems to which they are intended to connect.
19	Investigate software built-in extensibility features.
20	Review software complexity and look for alternatives to reduce the complexity.
21	Investigate the security of the data when passed from application servers to databases.
22	Avoid default or other improper configurations that may open the door to attackers.
23	Default to "highest security" needed, and require validation and approval for deviations.
24	Establish tools to be used for various stages of the life cycle that will be used for assessing security.
25	Perform security testing for unit and system integration.
26	Potentially, establish a security risk rating criteria and document the rating of the software product within the organization. Using a risk assessment tool can benefit this area.

Phase 2 of the SSC focuses on the external release of software (i.e., software that is developed for release external to the organizational environment). Prior to release, an evaluation/acceptance process is recommended. Figure 2 below depicts a potential process. The functions can be performed by the same role. However, it is preferable to keep the functions separate to avoid potential conflicts of interest. A description of roles and processes for Figure 2 is:

Developer: The developer creates the products to be released. (products include software, documentation, test data, configuration files, etc.) Products for release are sent to the Release Analyst function.

Release Analyst: This function analyzes the product created by the developer to determine if it meets the release criteria. All sensitive information in the product must have waivers, otherwise the product is sent back to the developer for rework.

Release Authority: The release authority receives the products to be released, any waivers and the Software Security Checklist(s). They approve the products for release.

Waiver Authority: The waiver authority issues waivers for any sensitive information allowed to remain in the release product. They may need to consult with the Security Authority on any given piece of sensitive information.

Security Authority: The Security Authority makes a determination if any piece of sensitive information can be waived.

Table 2 below provides the start of a sample set of issues and questions for external release of software.

IEEE COMPUTER SOCIETY

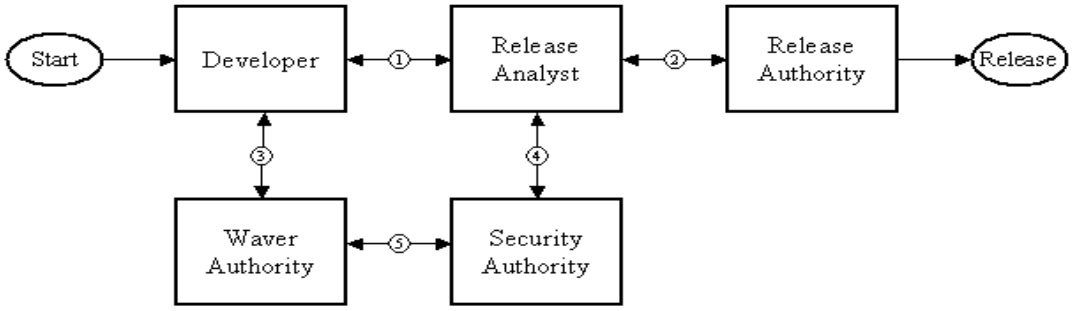

Figure 2: Software Release Functional Diagram

Table 2: Example of an Initial Start of a Security Checklist for the External Release of Software

1.0	Does the software include IP addresses and subnet ranges? 1.1 If yes, are these IP addresses sensitive? 1.2 Can these addresses be used to gain information that may pose a risk to the organization?
2.0	Does the software/program include Host names? 2.1 If yes, are these host names sensitive? 2.2 Does the release of these host names pose a risk to the organization?
3.0	Are there any settings that can be exploited? 3.1 If yes, can any of these settings be modified or deleted? 3.2 If settings can not be modified or deleted, would they pose a risk to the organization? 3.3 If settings can be modified or deleted, would they pose a risk to the organization?
4.0	Is there any non-sensitive information in the software that can be used to probe secrets? 4.1 If yes, can non-sensitive information be manipulated to expose sensitive information? 4.2 Can non-sensitive data be altered and modified so as to pose risk?
5.0	Is there any Material that might expose company information such as Customer lists? 5.1 If yes, are Customers lists protected under a privacy policy? 5.2 Does the release of Customers lists pose a risk? 5.3 Does the release of Customers lists do harm to the customers?
6.0	Is any of the data restricted? 6.1 If yes, is the data controlled by security mechanisms such as RBAC? 6.2 If yes, are their security restrictions on the transfer of restricted data? 6.3 Is the restricted data transmitted over open networks? 6.3.1 Is the restricted data encrypted before transmission?
7.0	. . .

3.1 Tools and instruments for use in the life cycle

The last item of the SSC addresses tools to assess security during the life cycle process. A number of tools are now available for use in the software life cycle that can be used to check and test system and software security beginning from the requirements phase through to the operation of the software. Many of these tools are from other formal disciplines such as reliability and safety and include modeling, code-auditing, fault/attack-trees and fault injection, property-based testing, boundary testing, etc. Current research in security modeling at JPL focuses on starting the software development process early in requirements by ensuring that there are no known violations of properties that lead to security weaknesses or defects.[15]

A security assessment instrument suite is valuable in producing secure software.[12] A taxonomy of security assessment tools is maintained by UC Davis. It discusses heir potential use in the life cycle along with alternate tools that may be used. It includes the relative strengths and weaknesses of these tools as well.[16]

IEEE COMPUTER SOCIETY

Additionally, they have offered a classification scheme for security tools.[17] It is hoped that this taxonomy will assist developers and analysts in producing more secure software.

The SSC should point to an SSAI for security tools that can be used to assess and mitigate security risks and exposures. It should include both the development and the maintenance phases of the software life cycle.

3.2 Maintenance and decommissioning of software

Maintenance and decommissioning are important but often forgotten aspects of the life cycle. The SSC should also cover these life cycle phases as well. Replacing or removing software should be carefully controlled to ensure that the rest of the system is not put at risk in the process. For example, the software may have installed some programs that it used, but constrained, and if the software is replaced or removed, the programs would be available for unconstrained use; or, it may have left behind logic bombs or other forms of Trojan horses. The SSC being developed will provide guidelines to cover this phase of the life cycle along with tools for regression testing of the software in its operating environment both. Maintenance is a critical phase of the life cycle as it covers more than 60% of the software life cycle process.

4. Conclusion

Integrating security in the software life cycle should be an end-to-end process beginning with a security risk analysis and requirements gathering, through design and development, testing and integration, include operations and maintenance, and finally decommissioning. To assist in the process, clearly stated corporate policies and requirements on security as well as guidelines are critical. Having a well-defined software security policy, risk rating for software and a checklist that supports the policy and requirements is essential as well.

An SSC for use in the life cycle along with tools to verify security will aid in producing more secure software and decrease risk to the corporate environment. With management support and a well-defined process, software security will no longer be an oxymoron, but will lead to more secure systems. An SSAI will provide ROI benefits to organizations that integrate it as part of their software development and maintenance life cycles.

5. Acknowledgements

The research described in this paper is being carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

6. References

[1] C. Mann, “Why Software Is so Bad,” Technology Review (July/August 2002).

[2] G. McGraw, “Software Assurance for Security,” IEEE Computer 32(4), pp. 103-105 (April, 1999).

[3] D. Gilliam, J. Kelly, M. Bishop, “Reducing Software Security Risk Through an Integrated Approach,” Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June, 2000), Gaithersburg, MD, pp.141-146.

[4] D. Gilliam, J. Kelly, J. Powell, M. Bishop, “Development of a Software Security Assessment Instrument to Reduce Software Security Risk” Proc. of the Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Boston, June 2001, pp 144-149.

[5] G. McGraw, “Building Secure Software: A Difficult But Critical Step in Protecting Your Business,” Cigital, White Paper, available at: http://www.cigital.com/whitepapers/.

[6] G. McGraw, “Software Risk Management for Security,” IEEE Computer, 32(4), April 1999 pp. 103-105.

[7] M. Bishop, Computer Security: Art and Science, Addison-Wesley-Longman, Nov. 2002.

[8] A. Jaquith, “The Security of Applications: Not All Are Created Equal,” Research Report, @Stake, February 2002, pp. 1-12, Internet Article: http://www.atstake.com/research

[9] K. Hoo, A. Saudbury and A. Jaquith, “Tangible ROI through Secure Software Engineering,” Secure Business Quarterly, Q4, 2001

[10] B. Robinson, ”Making Software NASA Tough,” Federal Computer Week, July 1, 2002

[11] D. Gilliam, J. Powell, J. Kelly, M. Bishop, “Reducing Software Security Risk Through an Integrated Approach”, IEEE Goddard 26th Annual Software Engineering Workshop.

[12] Component Based Model Checking, J. Powell, D. Gilliam, Proceedings of the 6th World Conference on Integrated Design and Process Technology, June 23-28, Pasadena CA, p66 & CD

[13] J. Viega and G. McGraw, Building Secure Software: How to avoid Security Problems the Right Way, Addison-Wesley, Boston, 2001.

[14] M., Howard and D. LeBlanc, Writing Secure Code, Microsoft Press, Redmond, WA, 2002.

[15] D. Gilliam and J. Powell, “Integrating a Flexible Modeling Framework (FMF) with the Network Security Assessment Instrument to Reduce Software Security Risk,” Proceedings on the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June 2002, pp. 153-160.

[16] UC Davis. A taxonomy of security tools and sites can be found at: http://seclab.cs.ucdavis.edu/projects/testing.

[17] M. Bishop, H. Briggs, E. Haugh, P. LeBlanc, “A Classification Scheme for Security Tools,” Internet Article: http://seclab.cs.ucdavis.edu/projects/testing/papers Accessed November 2002.

IEEE COMPUTER SOCIETY