SYM-AM-17-063

Proceedings
of the
Fourteenth Annual Acquisition Research Symposium

Wednesday Sessions
Volume I

Acquisition Research:
Creating Synergy for Informed Change

April 26–27, 2017

Published March 31, 2017

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

Acquisition Research Program
Graduate School of Business & Public Policy
Naval Postgraduate School

Security Measurement—Establishing Confidence That System and Software Security Is Sufficient

Carol Woody—PhD, has been a senior member of the technical staff at the Software Engineering Institute since 2001. She is the Technical Manager of the CERT Cybersecurity Engineering team which addresses security and survivability throughout the development and acquisition life cycles, especially in the early stages. Her research focuses on building capabilities for measuring, managing, and sustaining cybersecurity for highly complex networked systems and systems of systems. She coauthored *Cyber Security Engineering: A Practical Approach for Systems and Software Assurance*, published in November 2016 by Pearson Education, InformIT, as part of the SEI Series in Software Engineering. [cwoody@cert.org]

Abstract

Evaluating the software assurance of a product as it functions within a specific system context involves assembling carefully chosen metrics that demonstrate a range of behaviors to establish confidence that the product functions as intended and is free of vulnerabilities. The first challenge is to establish that the requirements define the appropriate security behavior and the design addresses these security concerns. The second challenge is to establish that the completed product, as built, fully satisfies the specifications. Measures to provide assurance must, therefore, address requirements, design, construction, and test. We know that software is never defect free. According to Jones and Bonsignour (2012), the average defect level in the United States is 0.75 defects per function point or 6,000 per million lines of code (MLOC) for a high-level language. Thus, software, on average, cannot always function perfectly as intended. Additionally, we cannot establish that software is completely free from vulnerabilities based on our research which indicates that 5% of defects should be categorized as vulnerabilities. So how can we establish reasonable confidence in software security? To answer this question, the Software Engineering Institute (SEI) is researching how measurement can be used to establish confidence in software security. This paper will share our progress to date.

Is the System Secure?

The Department of Defense (DoD) has followed a well-structured acquisition and development life cycle for decades. The rules for these activities are clearly laid out in Department of Defense Instruction (DoDI) 5000.2 (DoD, 2003)[1]. There are a series of activities and milestone reviews that require a program office to demonstrate to management and oversight groups that good engineering is underway as the acquisition of a system moves from an idea through to final implementation. Can we leverage this process focus to establish confidence that software assurance is well-integrated into each of the life cycle steps? The current acquisition focus is on establishing appropriate requirements and ensuring these are met in the delivery of the system.

The evaluation of the security of the same system is equally as well-structured using the guidance provided in DoDI 8510 (DoD, 2014). These guidelines were recently rewritten to broaden the focus beyond the security technology controls to include protection of critical

[1] The most recent version effective February 2, 2017, is available at http://www.acqnotes.com/wp-content/uploads/2014/09/DoD-Instruction-5000.02-The-Defense-Acquisition-System-2-Feb-17-Change-2.pdf

assets based on potential risks. If selected controls are implemented to protect these key assets and these are validated through the certification and accreditation reviews, is the system secure? It is hard to say since the engineers are focused on requirements and the security analysis is focused on security controls. How sufficient are the security requirements? Requirements are subject to change as the system progresses into the life cycle, but the security controls are not typically reviewed after the initial selection.

The acquisition is focused on delivery of a system, and software is typically viewed as a necessary component. In reality, software is quickly becoming the major component. Software handled 8% of the functionality of the F-4 Phantom fighter in 1960, but by 2000, its role grew to 80% for the F-22 Raptor, and the trend does not appear to be abating (National Research Council, 2010). We know that software is never defect free. According to Jones and Bonsignour (2012), the average defect level in the United States is 0.75 defects per function point or 6,000 defects per million lines of code (MLOC) for a high-level language. Very good code levels would be 600 to 1,000 defects per MLOC, and exceptional levels would be below 600 defects per MLOC. Thus, software, cannot always function perfectly as intended since there will always be defects. Additionally, we cannot establish that software is completely free from vulnerabilities based on our research which indicates that 5% of defects should be categorized as vulnerabilities (Woody, Ellison, & Nichols, 2014). Hence, as the role of software increases, the availability of software vulnerabilities also increases. It is impossible to avoid the constant news that systems are under attack and vulnerabilities are so prevalent that attackers are successful. Software security is a growing concern and needs to be effectively managed as part of an acquisition. Can we do this?

The Program Protection Plan[2] assigns a software assurance reporting responsibility to the program office, but they are typically not the group building and maintaining the software, so they pass this responsibility to the contractor. How will a program office know if there is sufficient software security in the system the contractor delivers? Few engineers in a program office are trained in security, and even with training, will they be able to directly evaluate the product? Maybe not, but they should be able to evaluate the quality of the processes used by the contractor in building, integrating, and verifying the system. Higher quality with fewer defects, along with a focus on software security, should result in fewer defects and fewer vulnerabilities.

The program office can ask the contractor to report on a wide range of metrics. There are metrics for cost, schedule, quality, complexity, resiliency, and technical debt just to list a few of the categories. Capers Jones (2015) reports that over 3000 different metrics are in use, and most of them are inconsistent at best and typically misleading. In his 2015 report, Jones notes “the software industry labors under a variety of non-standard and highly inaccurate measures compounded by very sloppy measurement practices.” What are the metrics for software security? We can count vulnerabilities just as we count defects. Can we assume all code will be of high quality so that defects and vulnerabilities are at a minimum? High quality development requires resources with the capability to deliver high quality as well as effective processes to ensure the results delivered meet expectations. What kind of useful measurements can we apply to these?

[2] See Defense Acquisition University (DAU; n.d.) for a description of the Program Protection Plan.

Using Engineering Evidence to Reduce Fear, Uncertainty, and Doubt (FUD) for Software

For a measurement to be useful in engineering, there are three questions that need to be addressed:

- Are we measuring the right things at the right time?
- Are our measurements trending in the right direction?
- Do we collect information soon enough to react to problems within other constraints?

As an example, let's consider how these questions are addressed in the acquisition of an airplane. How do we establish that the plane will fly when it is delivered? The planes are engineered to meet requirements that are defined for the expected use. The Dreamliner is designed to carry up to 330 passengers on long distances in comfort (Boeing, n.d.). It is reported to have a range of 11,910 kilometers, or 6,430 nautical miles. The wing span is 197 feet and 4 inches, height is 55 feet and 10 inches, and the length is 224 feet. All of these characteristics are determined based on lift and speed and other aerodynamic characteristics to allow the plane to meet its flight requirements. The F35 has a very different set of requirements as a single-seat, single-engine, all-weather stealth fighter plane (Lockheed Martin, n.d.). It is designed for a maximum speed of Mach 1.5 at altitude, with a range of approximately 1,620 nautical miles using internal fuel. The wing span is 32.78 feet, height is 13.33 feet, length is 50.5 feet, and wing area is 450 feet. Each of these planes is built in a highly structured manufacturing operation using best practices for constructing and testing the parts and validating the assembled whole. Reviews are conducted at scheduled times throughout the development and testing of prototypes is extensive to ensure requirements are met. There should be no surprises at the point of delivery about the plane's ability to meet its flight requirements (NASA, 2009). To apply this approach to software security, we need effective processes and a means of measuring how well they are working.

Building Blocks for Engineering Software Security

After determining that there were many software security practices, but nothing structured for evaluating software assurance, SEI undertook the task of developing the Software Assurance Framework (SAF)[3]. The SAF defines a set of cybersecurity practices that programs should apply across the life cycle and supply chain. The SAF can be used to assess a program's current practices to identify gaps and chart a course for improvement. By verifying and identifying improvements for a program's cybersecurity practices relevant to software, the SAF helps to (1) establish confidence in the program's ability to acquire software-reliant systems that are sufficiently secure, and (2) reduce the cybersecurity risk of deployed software-reliant systems. When developing the SAF, we leveraged the software acquisition and cybersecurity expertise of the SEI's technical staff and also referenced a variety of acquisition, development, process improvement, and cybersecurity documents including the following:

[3] A technical note, CMU/SEI-2017-TN-001, that provides a detailed description of the key practices selected for the SAF, is in the publication process and is expected to be available on the SEI website this spring.

- National Institute of Standards and Technology (NIST) Special Publication 800-53, titled *Security and Privacy Controls for Federal Information Systems and Organizations* (NIST, 2013)
- NIST Special Publication 800-37, titled *Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach* (NIST, 2010)
- DoDI 5000.2, titled *Operation of the Defense Acquisition System* (DoD, 2003)
- Capability Maturing Model® Integration (CMMI®; CMMI, 2007)
- Build Security In Maturity Model (BSIMM; McGraw, Migues, & West, 2015)

The selected practices fall into four general focus areas: process management, engineering, project management, and support. Within each of these general areas we have grouped practices within subcategories.

Process management would include the following categories of practices:

- process definition
- infrastructure standards
- resources
- training

Engineering would include the following categories of practices:

- product risk management
- requirements
- architecture
- implementation
- testing, validation, and verification
- support documentation and tools
- deployment

Project management would include the following categories of practices:

- project plans
- project infrastructure
- project monitoring
- project risk management
- supplier management

Support would include the following categories of practices:

- measurement and analysis
- change management
- product operation and sustainment

In order to link the detailed practices from the SAF framework to a specific program to address software assurance, we have used a standard software management technique, Goal-Question-Metric (GQM) Approach developed in the 1980s as a structuring mechanism (Basili, 1984). This is a well-recognized and widely used metrics approach. It requires the establishing of a goal for which we structure questions with associated metrics that begin to answer each question. In the following section, we will show an example of how these

building blocks can be used to structure an answer to our question about whether a system is secure.

Engineering for Software Security

If we apply the engineering approach described earlier about the plane's flying to security, we must start with establishing our engineering goal for software security. We would like to establish a requirement that software critical to mission and flight functions have no vulnerabilities, but this is not feasible (Woody, Ellison, & Nichols, 2014). However, we can structure a goal for the airplane that the critical software be of the highest quality such as:

> Mission- and flight-critical applications executing on the plane or used to interact with the plane from ground stations shall be high quality, with no more than 600 defects per MLOC and vulnerability levels below 30 per MLOC.

As the saying goes, quality cannot be tested in, it must be built into the product (Koch, n.d.). In order to meet this goal, the contractor would have to ensure they are building the system using high quality engineering processes at each step of the life cycle. This system goal would need to flow down to each step in building the software. To define how we might measure and monitor these processes to ensure high quality, we can create sub-questions for each major area of software development that reflect the contribution to be made to the system as follows:

- **Software Requirements.** Does the program/project define and manage software security requirements?
- **Software Architecture**. Does the program/project appropriately address security in its software architecture and design?
- **Implementation**. Does the program/project minimize the number of vulnerabilities inserted into the code?
- **Testing**, **Validation**, **and Verification**. Does the program/project test, validate, and verify security in its software components?
- **Support Tools and Documentation**. Does the program/project develop tools and documentation to support secure configuration and operation of software components?
- **Deployment**. Does the program/project consider security during the deployment of software components?

Each of these questions could be addressed within the software life cycle through the selection of appropriate practices, outputs, and metrics that demonstrate quality results. For each of the software development areas, we would want to confirm that best practices are performed and these are producing expected outputs along with metrics appropriate to expected results.

As an example, since much of the concern with software security is tied to vulnerabilities, consider how we could be confident that the number of vulnerabilities introduced into the critical code are minimized. There are several best practices in secure coding that we would expect to be performed as follows:

- Secure coding standards are applied.
- Code developers are trained in the use of secure coding standards.

- Evaluation practices (e.g., code reviews and apply tools) are applied to identify and remove vulnerabilities in delivered code (including code libraries, open source, and other reused components).

In addition, we should expect to see outputs and metrics that reflect that these practices are appropriately addressed as shown in Table 1.

Table 1. Activities/Practices, Outputs, and Metrics

Activities/Practices	Outputs	Candidate Metrics
Secure coding standards are applied	Policy that requires the use of secure coding standards Contract language to ensure vendor(s) practices require use of secure coding standards	% of vendor contracts including requirements for the use of secure coding standards % of system developed using secure coding standards % of code verified for secure coding standard conformance
Code developers are trained in the use of secure coding standards	Competency standards for code developers require training in secure coding standards Hiring qualifications require training in secure coding standards Contract language requires use of developers trained in secure coding standards	% of software developers trained in secure coding standards % of code supported by developers trained in secure coding standards
Evaluation practices (e.g., code reviews and apply tools) are applied to identify and remove vulnerabilities in delivered code (including code libraries, open source, and other reused components)	Policy that requires the use of evaluation practices to identify and remove vulnerabilities and reporting of metrics Output of evaluations Corrections documented Contract language requires use of evaluation practices to identify and remove vulnerabilities and metrics reporting	% of vendor contracts requiring use of evaluation practices and reporting of vulnerability metrics Code coverage: % of code evaluated (total and by each type of review) Vulnerabilities per MLOC identified and removed Unaddressed vulnerabilities per MLOC % code libraries evaluated % open source evaluated % legacy components evaluated Count of high priority vulnerabilities identified and count of those removed

Subsequent steps in the development process should continue to confirm that vulnerabilities are at a minimum through testing, validation, and verification.

Conclusion and Next Steps

Returning to our initial question about determining if the system is secure, we have established that our evaluation must focus sufficient attention on the quality and security built into the software which makes up over 80% of the functionality of most systems. If this software is well-defined, well-built, and well-implemented using best practices for engineering software with good security, we should be able to review outputs and confirm through metrics with reasonable confidence that the system is secure. There is information we can collect and evaluate all along the life cycle about the product, processes, and practices to give us confidence in achieving our goal that the final product will be sufficiently secure. System engineering reviews can be used to confirm progress as follows:

- **Initial Technical Review (ITR).** Assess the capability needs (including software security) and materiel solution approach.
- **Alternative Systems Review (ASR).** Ensure that solutions will be cost-effective, affordable, operationally effective. Ensure that solutions can be developed in a timely manner at an acceptable level of software security risk.
- **System Requirements Review (SRR).** Ensure that all system requirements (including security) are defined and testable, and consistent with cost, schedule, risk (including software security risk), technology readiness, and other system constraints.
- **Preliminary Design Review (PDR).** Evaluate progress and technical adequacy of the selected design approach including software security.
- **Critical Design Review (CDR).** Determine that detail designs satisfy the design requirements (including software security) established in the specification and establish the interface relationships.

References

Basili, V. R., & Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering, SE-10*(6), 728–738.

Boeing. (n.d.). Boeing 787. Retrieved from http://www.boeing.com/commercial/787/#/by-design

Chrissis, M. B., Konrad, M., & Shrum, S. (2011). *CMMI for development: Guidelines for process integration and product improvement* (3rd ed.). Boston, MA: Addison-Wesley.

Defense Acquisition University (DAU). (n.d.). *Program protection plan*. Retrieved from https://acc.dau.mil/CommunityBrowser.aspx?id=549148&lang=en-US

DoD. (2003). *Operation of the defense acquisition system* (DoDI 5000.2). Retrieved March 20, 2017, from: http://www.acq.osd.mil/dpap/Docs/new/5000.2%2005-12-06.pdf

DoD. (2014). *Risk management framework for DoD information technology* (DoDI 8510.01). Retrieved from http://www.dtic.mil/whs/directives/corres/pdf/851001_2014.pdf

Jones, C. (2015, January 26). *The mess of software metrics V5.0* (Namcook Analytics).

Jones, C. & Bonsignour, O. (2012). *The economics of software quality* (Addison Wesley).

Koch, A. S. (n.d.). Testing is not quality assurance. PMP. Retrieved from http://www.projectconnections.com/articles/080306-koch.html

Lockheed Martin. (n.d.). F-35 Lightning II. Retrieved from https://www.f35.com/about

McGraw, G., Migues, S., & West, J. (2015). Building Security in Maturity Model (Version 6). Cigital. Retrieved October 10, 2016, from https://www.bsimm.com/

NASA. (2009). *Final report: NASA study on flight software complexity*. Retrieved from http://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf

National Institute of Standards and Technology (NIST). (2010). *Guide for applying the risk management framework to federal information systems: A security life cycle approach* (NIST Special Publication 800-37, Rev. 1). Retrieved from http://csrc.nist.gov/publications/nistpubs/800-37-rev1/sp800-37-rev1-final.pdf

National Institute of Standards and Technology (NIST). (2013). *Security and privacy controls for federal information systems and organizations* (NIST Special Publication 800-53, Rev. 4). Retrieved from http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

National Research Council. (2010). *Critical code: Software producibility for defense*. Retrieved from http://www.dtic.mil/dtic/tr/fulltext/u2/a534043.pdf

Woody, C., Ellison, R., & Nichols, W. (2014). *Predicting software assurance using quality and reliability measures* (CMU/SEI-2014-TN-026). Retrieved April 2, 2017, from the Software Engineering Institute, Carnegie Mellon University website: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589

Acknowledgments

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004623

Acquisition Research Program
Graduate School of Business & Public Policy
Naval Postgraduate School
555 Dyer Road, Ingersoll Hall
Monterey, CA 93943

www.acquisitionresearch.net