Minimum standards for vendors' testing of their software source code

Discussion: The majority of this paper has been extracted from a presentation given to a member of the Senate Armed Services Committee staff in support of Public Law 112-239, National Defense Authorization Act for Fiscal Year 2013, section 933, Improvements in Assurance of Computer Software Procured by the Department of Defense. The presentation focused on five different areas where vendor testing of software source code could/should be done together with a set of suggested deliverables. In addition, a simplistic business case that determined the most effective time to test the source code with the resulting cost benefits. The business case model was provided to one of the major tool vendors who used it as part of their successful delivery to a number of major commercial financial and industrial civilian corporations. Most of the details were not new and, in fact, have not changed in the past decade. What has changed is the sophistication of the threat and the need to move much more swiftly and correctly to counter. Regardless, the one thing that has remained constant is to scan the source code early and fix those discovered errors before the code is promoted for testing and/or use.

The presented approach to testing source code was called at the time, and is still called today, Software Code Quality Checking (SCQC). The approach was developed by the late Dr. Gregory Guernsey and implemented by the author of this paper. It was based on a multi-disciplined, multi-deliverable methodology. Deliverables can provide guidance as to the design of the software system but, more importantly, should be used to capture decisions made.

Dr. Guernsey defined SCQC as: "a scan of the source code, executables, and related artifacts, e.g., documentation, to ensure that the System Under Review can continue with development, demonstration, and test; and can meet the stated performance, maintainability, and usability requirements within cost (program budget), schedule (program schedule), risk, and other system constraints. It is a variation on the legally approved definition for DOD for software assurance as: "the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the life cycle." Of course, both definitions are inconsistent with but conceptually equivalent with the seven or eight other definitions of software assurance one can easily find on the web.
ACTION ITEM 1. Eventually settle on one consistent and complete definition of Software Assurance.

Dr. Guernsey decomposed SCQC into five (5) components defined here:

1. **Static Analysis** is the analysis of computer software and related documentation that is performed without actually executing programs built from the software. It is a recognized BEST practice that SHOULD precede Static Security Analysis.

2. **Static Security Analysis** is the analysis of computer software that is performed without actually executing programs to detect and report weaknesses that can lead to security vulnerabilities.

3. **Dynamic Program Analysis** is the analysis of computer software and related documentation that is performed by executing programs built from that software on a real or virtual processor.

4. **Dynamic Security Analysis** is the analysis of computer software that is performed by executing programs to detect and report weaknesses that can lead to security vulnerabilities.

5. **Architectural Analyses** may be supported by automated tools but are usually conducted by manual walk-through of documentation and visual inspection of the code.

These five (5) represent the ideal state for testing source code but, due to cost and schedule constraints, all 5 were rarely executed. Components 1, 2 and 5 are the minimum standards with items 3 and 4 done as time permits.

The definitions can be seen to be somewhat in conflict with some of the terminology in use today. Security scanning tools are often called static analysis tools when they are in fact static security analysis tools. Calling them static analysis tools disguises the importance of scanning the source code for TECHNICAL weaknesses first. It has been shown that Bad Coding Practices produce TECHNICAL weaknesses that, many of which, lead to SECURITY Weaknesses and, by focusing on writing technically correct code, many security weaknesses will be prevented, thus leaving very little for the Static Security Analysis Tools to discover.

During the early days of performing SCQC, static security analysis tools were not immediately available so the emphasis was put on RUTHLESS enforcement and INDEPENDENT AUDITING of the source code for technical correctness by the development teams. The audited code was turned over to the independent static security analysis team and, using the appropriate tools, NO critical or high risk security weaknesses remained after a few review cycles. The few medium and low risk were recommended for burn-down over time.

A more recent analysis concluded that 556 security weakness "types" identified by one tool's taxonomy related to 6 different technical weakness types that included availability, maintainability, safety (MISRA), maintainability, bad coding practices and performance. A second analysis that mapped security weaknesses to eight (8) other factors found just one (1) security "type" with just two (2) findings that were not addressed by the 8 "non-security" findings reported by the tool. This provides two paths to removing security weaknesses from your source code. Do you focus on quality first, anticipating that MANY security weaknesses will be de facto corrected or do you focus on security weaknesses and assure that technical weaknesses will be corrected as well? The choice will depend on your current business needs as well as compliance with the EO.

As indicated, an Architectural assessment and manual review of the code should also be performed to see how the components fit together and whether or not additional weaknesses have been introduced due to the limited type of coverage provided by the static code and static security analysis tools.

IN SUMMARY: Vendor testing of source code is but one piece in the overall Cybersecurity puzzle. Doing it correctly will remove a significant amount of the clutter we see in many current applications which CAN provide an easier path to seeing any dangerous security issues in the code. Get rid of the easy things first.

Cybersecurity is a JOURNEY, not a destination. It is rarely DONE because new threats and new rules consistently show up. In some cases, the rules negate things that many assumed to be correct a few years ago. Expect change and adjust accordingly.

And finally, several Defense Acquisition University courses contain a slide that states "A fool with a tool is still a fool." Creation of user guides and user training is part of the process. When properly trained in the use of the selected tool, the developers tend to instinctively begin writing code that complies with the rules.

Prologue: Public Law 112-239, National Defense Authorization Act for Fiscal Year 2013, section 933, Improvements in Assurance of Computer Software Procured by the Department of Defense was preceded by Public Law 111-383, Ike Skelton National Defense Authorization Act for Fiscal Year 2011, section 932, Strategy on Computer Software Assurance and followed by Public Law 113-66, National Defense Authorization Act for Fiscal Year 2014, section 937, Joint Federated Centers for Trusted Defense Systems for the Department of Defense. The series of laws were intended to change the paradigm for checking/testing/validating source code to one where automated tools became the primary means of accomplishing the task SUPPLEMENTED by manual review to fill in areas where the tools provided inadequate coverage. The presentation was reviewed and approved by the major command as well as the offices of the DOD CIO and the USD (AT&L) prior to delivering it to the Senate Staffer. Readers of this paper may find a more detailed study of the sections of the three Federal Laws cited above to see what else was said about scanning software source code.