Suggested Language to Incorporate Software Assurance into Department of Defense Contracts

Department of Defense (DoD) Software Assurance (SwA) Community of Practice (CoP) Contract Language Working Group

February 2016

DoD SwA CoP Working Group

- Chair: John R. Marien
- Co-Chair: Robert A. Martin

Abstract

This document is intended as a source for a web available resource setting forth a discussion on how to guide the application of Software Assurance concepts and deliverables to a new or existing contract for the delivery of software that behaves securely and is resilient to hazards and attack with respect to fulfilling the mission that the software supports. The examples cited in this document are not intended as copy-and-paste examples but rather a collection of examples that may be leveraged. Ideally, Software Assurance language should be included in the RFP. However, since there are many program already under contract, this document also addresses how and where to apply software assurance contract language throughout the software development lifecycle.

Table of Contents

List of Figures

List of Tables

1 Introduction

Common industry practice and Section 933 of the National Defense Authorization Act for Fiscal Year 2013 (Public Law 113-239) [1], define "software assurance" to mean 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. This document suggests language that may be tailored for use in Request for Proposal (RFP) packages and contracts to provide a government program office insights into the software development activities of its contractors and to provide assurance regarding developed software and its ability to meet the mission needs. With so much of today's mission functionality realized through software, both the traditional types used for planning, management, and logistics, as well as that used directly in weapon systems, vehicles, infrastructure management and utilities, the need for assurance about that software continues to grow. This language will generally appear in Sections C, I, L, and M of the standard format RFP and contract.

The following sections take a closer look at how RFP and contract language can be used to effectively address the three key concepts of this definition: *confidence* in a system, that the system *functions as intended*, and that the system is *free of vulnerabilities*.

The Department of Defense (DoD) Program Protection Plan (PPP) Outline and Guidance [2] Software Assurance Table [2, 3] includes a variety of measures and measurements, described and more fully explained in Section 13.7.3 of the Defense Acquisition Guide (DAG) [4], that focus on these three key concepts of conducting software assurance for a system.

1.1 Confidence

Confidence regarding contractors' assurance activities comes from obtaining appropriate information that a program office and others can understand and that supports the claims about the functionality of the software as well as the addressing of exploitable constructs in the system. The use of standardized collections of weaknesses, vulnerabilities, and attack patterns makes the understanding of contractors' assurance actions easier and more consistent and offers opportunities for reuse of that approach to provide similar confidence in other system needed and other contractors. There are several points in the lifecycle of a system when insights into the risks and the mitigation of those risks can be obtained and the Program Protection Planning (PPP) Software Assurance Table calls out several of them explicitly, as shown in Figure 1, so that the appropriate information can be collected and reviewed at the earliest stage of development supporting risk management decision making.

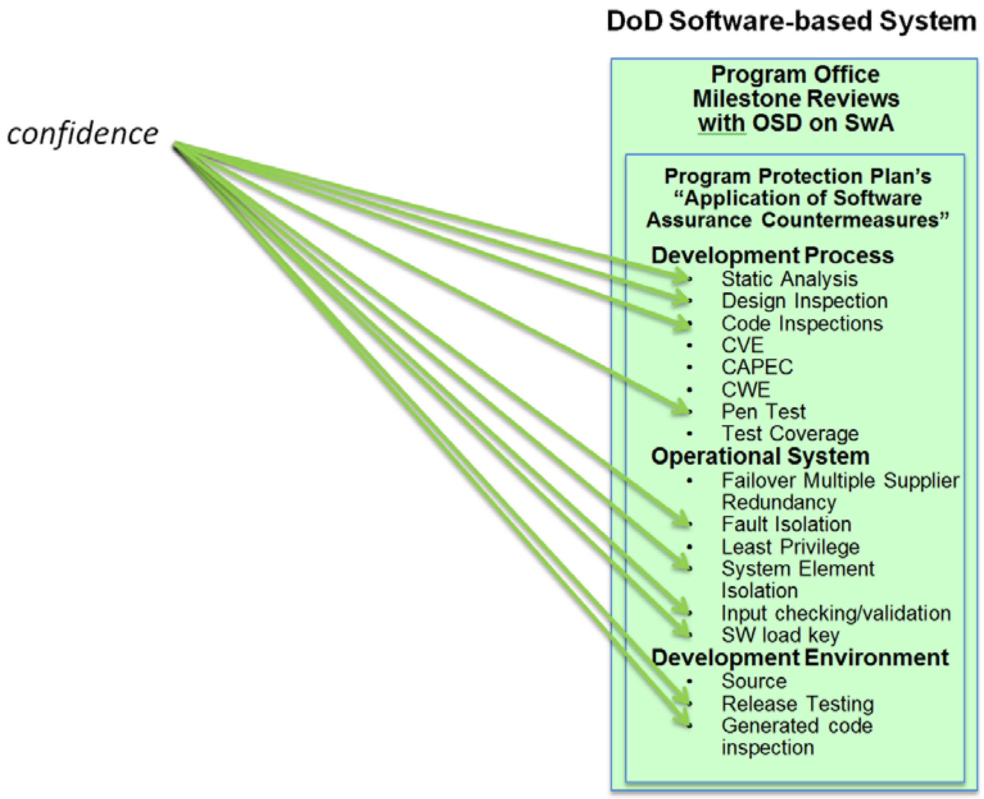

DoD Software-based System
confidence
Program Office Milestone Reviews with OSD on SwA
Program Protection Plan's "Application of Software Assurance Countermeasures"
Development Process
Static Analysis
Design Inspection
Code Inspections
CVE
CAPEC
CWE
Pen Test
Test Coverage
Operational System
Failover Multiple Supplier Redundancy
Fault Isolation
Least Privilege
System Element Isolation
Input checking/validation
SW load key
Development Environment
Source
Release Testing
Generated code inspection

Figure 1: Confidence

We build confidence in the software system through *static analysis*, *design inspection*, *code inspections*, and *penetration testing* during the development process. Each of these activities identifies weaknesses and vulnerabilities in the software and its design and allows easy, and early, correction at minimal cost and time.

1.2 Functions as Intended

Determining whether a system "functions as intended" requires both showing through testing that the intended functionality is there and understood by test coverage metrics and understanding what testing effort was made to make the system perform functions it is not supposed to do. As with the confidence measures discussed above, there are several points where insights into the risks and the mitigation of those risks regarding a system's ability to "function as intended" can be obtained and the PPP Software Assurance Table calls out several of them explicitly, as shown in Figure 2, so that the appropriate information can be planned for, collected, and reviewed at the appropriate stage of the System Development Life Cycle (SDLC). The Common Attack Pattern Enumeration and Classification (CAPEC) [5, 6] contains a collection of patterns of attacks that can be used to describe misuse and abuse testing done on a system either through automated tools or through pen test team activities.

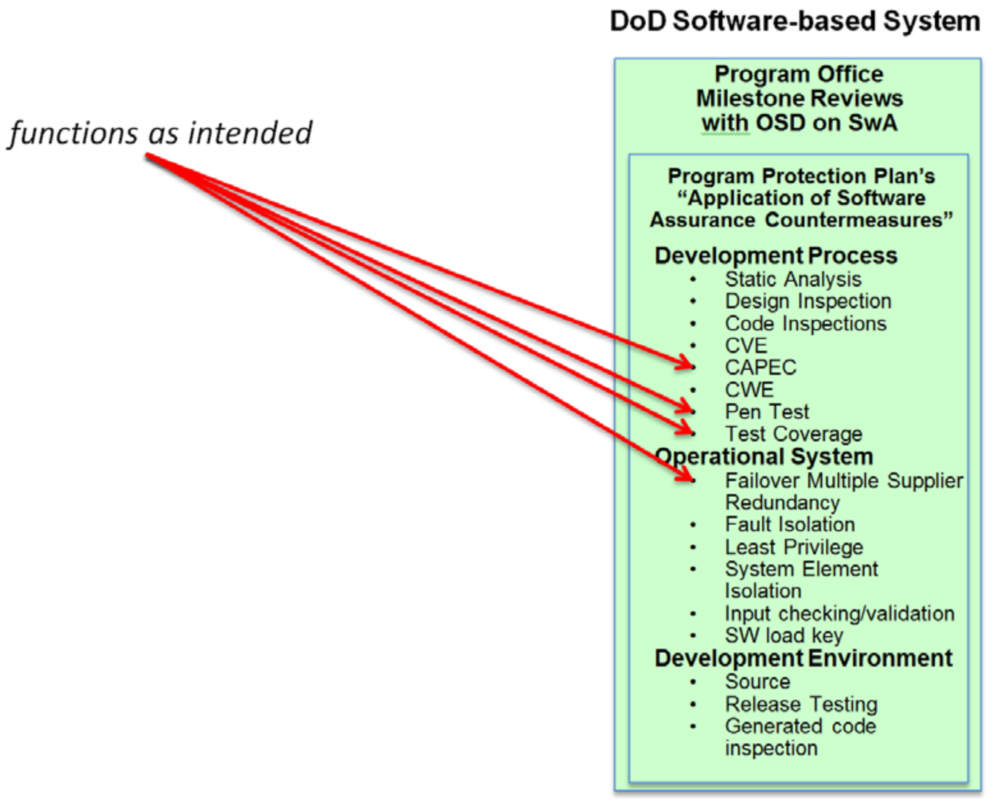


Figure 2: Functions as Intended

We assert that the software *functions as intended* through application of *CAPECs*, *penetration testing*, *test coverage*, and through *failover multiple suppliers*. Each of these activities allows us to identify that the software is indeed functioning as intended without functioning, or being made to function, in unintended ways.

1.3 Free of Vulnerabilities

It is common practice in industry [7, 8, 9, 10, 11, 12, 13, 14, 15, 16] in assessing whether software is "free of vulnerabilities" to refer to the Common Weakness Enumeration (CWE) [17, 18, 19] catalogue. This approach allows others to understand both what was "looked for" and what wasn't but could have been "looked for".

Similarly, for commercial software packages (proprietary and open source) being used as part of a system, the collection of publicly known vulnerabilities in these types of software, called the Common Vulnerabilities and Exposures (CVE) [20, 21] dictionary, is almost always used a reference source to determine if known issues have been mitigated.

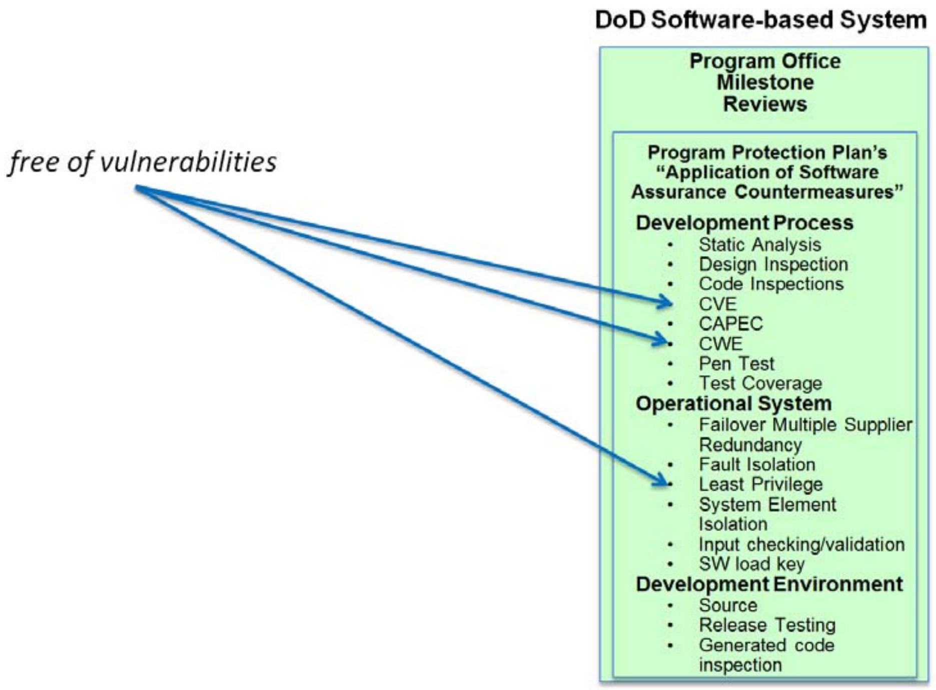


Figure 3: Free of Vulnerabilities

We assert that the software is *free of vulnerabilities* by validating that those CVE and CWE items that are most dangerous to the mission, are absent from the software and that the software operates at the *least privilege* required to complete its task.

Figure 3, above, shows how the listed types of information can give insights into activities and their results with respect to determining whether a system is "Free of Vulnerabilities" as outlined in the PPP Software Assurance Table and explained in DAG section 13.7.3. Of course, getting these types of findings and informational items for a particular system during its design, development, test, and post-deployment sustainment activities will require that the contract provide for the contractor to collect the necessary information or provide the government the opportunity to collect it directly.

Different parts of a contract will guide or require these different artifacts and activities from a contractor and it is critical that the contract fully describe what is expected from the contractor's development activity. The System Performance Specification, the Statement of Work, System Engineering Plan, Incentive Plan, CDRLs and other parts of a contract all have their appropriate part in guiding the expectations and performance of the developing organization and ensuring that they and their DoD customer understand each other's expectations and concerns but these activities are all predicated on the government having access to the custom and reused software for independent inspection, testing, and evaluation, so including a clause about delivery of the source code, all libraries, and frameworks is crucial.

A notional Statement of Objectives (SOO), as shown in Figure 5, could cover software assurance related activities for the Software Development, the Software Interfaces, as well as the use of Software Tools and Metrics. Working from the Systems Description document we can map these concepts to typical contract sections as illustrated in Figure 4 below.

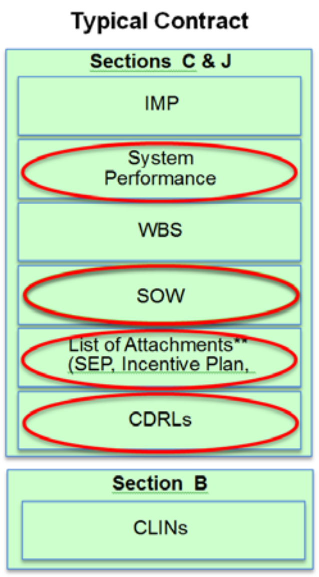


Figure 4: Typical Contract Structure

Using the SOO as shown as an example in Figure 5, we can identify notional SOO items related to Software Assurance.

Software Development:
It is the Government's objective for the contractor to develop secure, reliable, resilient, assured software:

- The contractor shall develop a secure coding guide which specifies language, required constructs/practices/patterns, prohibited constructs/practices/patterns, software comment requirements
- The contractor shall implement formal (e.g., Fagan) design and code inspections
- The contractor shall verify all code against the CWE, CVE, OWASP
- The contractor shall perform an origin analysis of all third-party libraries and frameworks used and ensure the versions used in the delivered software have no publicly known vulnerabilities and continue to review for newly reported vulnerabilities throughout the sustainment of the delivered software
- The contractor shall ensure all developers are trained and held accountable for development of secure code

Software Interfaces:
It is the Governments objective for all software interfaces to be completely documented and utilize strong typing and range checking

Software Tools:
It is the Governments objective for the contractor to utilize automated tools to maximize efficiency and effectiveness:

- The contractor shall utilize automated tools to support software configuration control
- The contractor shall utilize automated testing tools to support regression testing for all custom code with at least 90% statement coverage.
- The contractor shall utilize at least 2 different commercial static analysis tools to measure software quality and access vulnerabilities. Multiple tools are to be used to improve detection of software quality issues and vulnerabilities as each tool uses different detection methods.

Software Metrics:
<etc.>

Figure 5: Notional SOO Items

The specific wording for each of the above notional SOO items should be adjusted as required for a specific contract.

2 Inserting Software Assurance into Contract Language

Figures 6 through 9 illustrate that specific items that need to be in your contract to guide the various software assurance activities must be inserted into the contract during the contracting process. This is important because software assurance must be applied throughout the lifecycle of the software and must be addressed in contract vehicles from the RFP through sustainment activities.

If you have an existing contract, which is already in development, or are in sustainment, and need to add software assurance items, these changes will need to be inserted into the appropriate parts of the existing contract as illustrated in Figure 6 by a bilateral contract modification agreed to by the contractor. Adding these types of risk reduction activities and measurement opportunities to an existing contract will be disruptive and costly. The risk to the operational system posed by the unknown and unexpected vulnerabilities and frailties must be considered against the potential cost and perturbation to the project of identifying and removing them.

Whether you have an existing contract or are in the process of establishing one as shown in Figures 7 through 9, the allocation of liability for software defects and vulnerabilities must either be directly put in the contract through the mechanisms described in the subsequent parts of this document and the referenced examples, or you can try to rely on normal liability protections from either the Uniform Commercial Code (UCC) if you can make the case that the software in question is "goods" versus a "service" and the developer may be subject to suit under a strict liability theory of tort for residual vulnerabilities and susceptibility to hazards and attacks.

The specifics of the program office's concerns about liability have to be explicitly described, and discussed, so that both the developer and the government have a clear understanding of what is and is not the result when flaws are found in the operational system and fixes are needed or the impact from a failure needs to be addressed.

For those interested in understanding the general liability issues surrounding software, a good starting point on the question of contractor liability for software issues is the May 26, 1990 article posted at the Berkeley Technology Law Journal (BTLJ) in Volume 5 entitled "Software Product Liability: Understanding and Minimizing the Risks and Michael D. Scott, Tort Liability for Vendors of Insecure Software: Has the Time Finally Come?" [22].

The question of whether the development team is liable for fixing the residual issues in their software, or if they hold liability for the damage or fall-out from the vulnerable and/or unreliable nature of the software allowed to happen or caused, is still open. Until such time as severe monetary awards are levied against software vendors/developers who release faulty code there will be no incentive for software vendors/developers to build security into their software. The only reasonable alternative is to explicitly state what faults are to be repaired by the software vendor/developer in the contract agreed to by both parties.

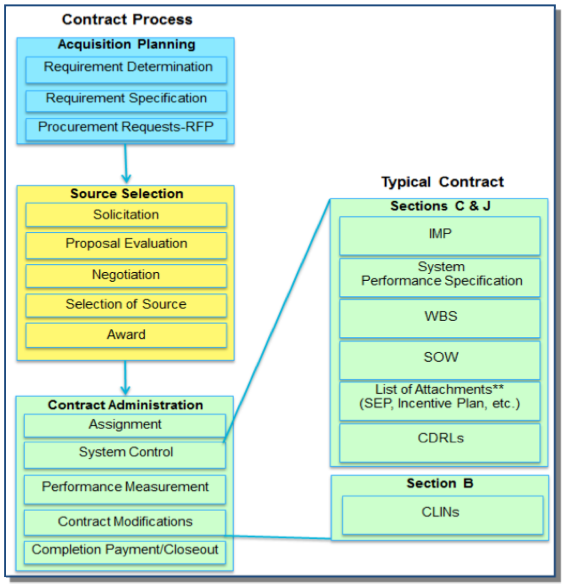
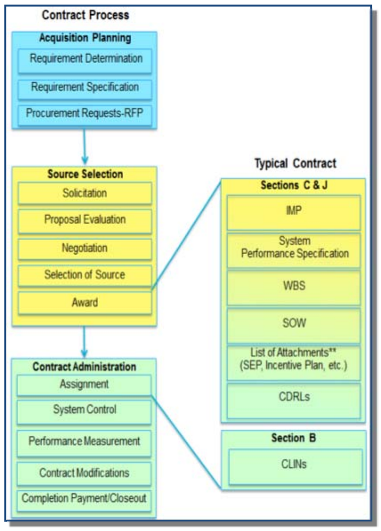
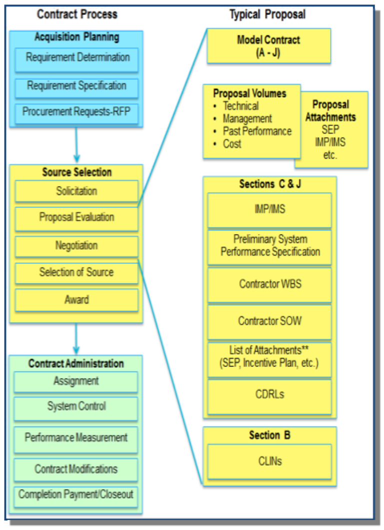


Figure 6: Contract Process for Existing

In a new procurement the program office will need to make sure that the appropriate parts of the contract have the right language. To do that you will have to make sure that the Source Selection process as shown in Figure 7 results in a contract with the needed language in the right parts. To do that, the appropriate language needs to be part of the selected proposal or proposals, as shown in Figure 8, which means it needs to be included in the RFP and addressed by the criteria used to evaluate those proposals, as shown in Figure 9.





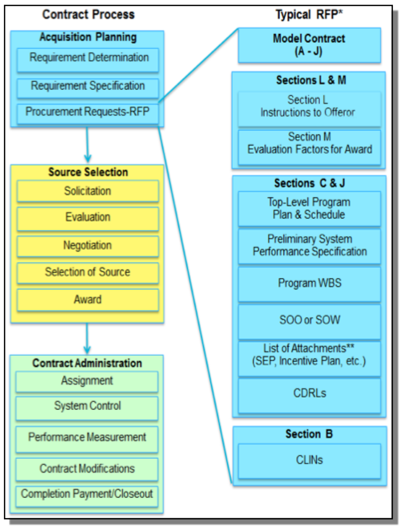


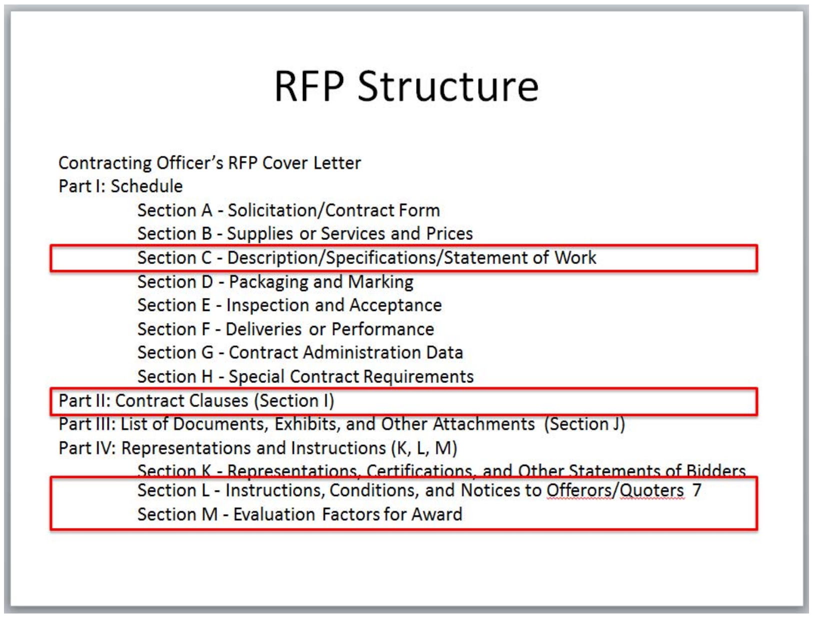
Figure 7: Contract Process

Figure 8: Contract Process - During Proposal

Figure 9: Contract Process - Request For Proposal (RFP)

For a new procurement to incorporate the appropriate software assurance activities into the contract there needs to be a clear articulation by the government about what is required. Additionally a clear statement that the requirements for these aspects of quality and their verified presence in the software must be a part of the selection criteria in the final award, part of the source selection process, and part of the performance criteria and incentive plan for the awarded contract.

Within the traditional RFP package, the appropriate language to be inserted goes in Section C, Section I, and Sections L & M, as highlighted in Figure 10.



RFP Structure

Contracting Officer's RFP Cover Letter
Part I: Schedule
    Section A - Solicitation/Contract Form
    Section B - Supplies or Services and Prices
    Section C - Description/Specifications/Statement of Work
    Section D - Packaging and Marking
    Section E - Inspection and Acceptance
    Section F - Deliveries or Performance
    Section G - Contract Administration Data
    Section H - Special Contract Requirements
Part II: Contract Clauses (Section I)
Part III: List of Documents, Exhibits, and Other Attachments (Section J)
Part IV: Representations and Instructions (K, L, M)
    Section K - Representations, Certifications, and Other Statements of Bidders
    Section L - Instructions, Conditions, and Notices to Offerors/Quoters 7
    Section M - Evaluation Factors for Award

Figure 10: RFP Structure

The subsequent example clauses and other procurement language examples target the above mentioned RFP Section by name, for example the Air Force Life Cycle Management Center (AFLCMC), AFLCMC/EZC - "Engineering Model RFP" (EM RFP) [23] language package provides modules with suggested items for several sections of an RFP to address different issues areas. These areas include modules on Software Assurance, Software Engineering, Software Metrics, and Supply Chain Risk Management (SCRM), Independent Verification and Validation, Quality Assurance, System Security Engineering, Configuration Management, Anti-Tamper, Firmware, and one on concerning Internal and External Interfaces.

Some of the examples provided do not offer specific clauses but rather discuss the issues that need to be addressed, while others are examples of existing contract clauses that are in use by components of the DoD or other government agencies to address some software assurance issues.

The important point is to remember that if you want a contractor to follow a particular approach or report and discuss issues dealing with software assurance in a particular manner, specify that approach or report and require the discussion of those issue in the contract, deliverables, and incentive plan. In order to accomplish that, we must show to the contractor that the government values quality and assurance by explicitly tying the delivery of quality software and software free of vulnerabilities to financial rewards and delivery criteria.

We must not sacrifice critical aspects of quality for reduced cost or improved schedule when those diluted requirements put the operational integrity of the software in question. In order for software to support the mission(s) it is planned to support, we must give the contractor good technical requirements (e.g., resilient design), require good practices (e.g., use code inspections), structure deliverables to require reporting issues in a standards-based manner (with CVEs, CWEs, and CAPECs), and support quality both initially at the time of delivery and in operation and sustainment.

2.1 Section I clause example

A sample Section I clause would include instructions to the offeror to require all Software Assurance requirements applied to the Prime contractor are also applied by the Prime contractor to all sub-contractors. Example entries for section I could be:

Section I:

- The Software Assurance requirements detailed in this RFP shall apply to all of the software delivered by the Prime contractor. Should the Prime contractor employ sub-contractors, then the Prime contractor shall require all of the Software Assurance requirements detailed in this RFP from each sub-contractor they employ for use on this contract.
- All software assurance requirements defined herein shall apply to all reused code included and delivered by the Prime contractor and its sub-contractors.
- All software assurance requirements defined herein shall apply to all reused objects, both proprietary and open source, and their originating reused source code, whether that code was included and delivered by the Prime contractor and its sub-contractors or just referred to by them as the source of the reused object.
- The government will provide to the Prime contractor a list of the Top-*n* most important CWEs. All software delivered by the Prime contractor shall be free of all of the CWEs in the government's Top-*n* CWE list. If a CWE in the government's Top-*n* CWE list is found to be present in the delivered software, the Prime contractor shall be liable only to repair the defect within XX-days at the contractor's expense.

2.2 Section L&M clause examples

It is important that for each instruction, condition, and notice to offerors included in Section L of an RFP that there be a corresponding paragraph in Section M indicating how to evaluate the item from Section L. An example entry for section L is:

Section L: Paragraph XX: Software Assurance Sample Report:

- The Offeror shall provide a report titled "Software Assurance Analysis"
- The "Software Assurance Analysis" report shall contain two sections:
  - Section 1: Details how static analysis for Software Assurance is used within the offeror's development lifecycle in writing and in diagrammatic form for illustrative purposes
  - Section 2: Contained within a spreadsheet file, details the output from a minimum of two software static analysis tools used by the offeror according to the following table (not including the two samples provided):

CWE #	Severity or Grade	Violation Description	Violation Location	Tool Name	Notes (optional)
CWE-89	1	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	C:\Sample\Project\XYZZY.cs [line:23]		
CWE-481	2	Assigning instead of Comparing	C:\Sample\Project\XYZZY.cs [line:203]		

- The "Software Assurance Analysis" report shall provide a static analysis of the legacy (GFI) code provided <*make this section as specific as possible to include file names or code groups where possible*>
- The "Software Assurance Analysis" report shall have a primary sort on Column 1 so that all **CWE-###** entries are at the top of the report in ascending order by CWE #
- The "Software Assurance Analysis" report shall have a secondary sort on Column 2 so that the highest priority results are at the top of the report

Section M: Paragraph XX: The government has run a static analysis tool on the specified code to serve as a baseline to spot-check known violations in the sample source code provided by the government as part of the bidder's library.
The Software Assurance report will be graded as follows [100 points total + bonus points]:

- Report named correctly [10 points]
- Report formatted correctly [10 points]

- Report covers the specified source code from **Section L paragraph 23** <n*ote to reader: this Section M item corresponds exactly to the specified Section L item*> and provides text describing detailing how, and in which part(s) of the software development lifecycle, static analysis for Software Assurance is used [10 points]
- Report entries include CWE numbers [60 points]
    - 1-5 unique CWE entries [10 points]
    - 6-10 unique CWE entries [20 points]
    - 11+ unique CWE entries [30 points]
- Report entries include non-CWE labeled violations [10 points]
- Bonus: 10 points if each of the following 10 CWEs are identified in the sample source code. Note that there is no guarantee that the following CWEs are present in the sample source code: *<note to reader: This is a good place to insert your Top-N CWE list>*
    1. CWE-xxx
    2. CWE-xxx
    3. CWE-xxx
    4. CWE-xxx
    5. CWE-xxx
    6. CWE-xxx
    7. CWE-xxx
    8. CWE-xxx
    9. CWE-xxx
    10. CWE-xxx

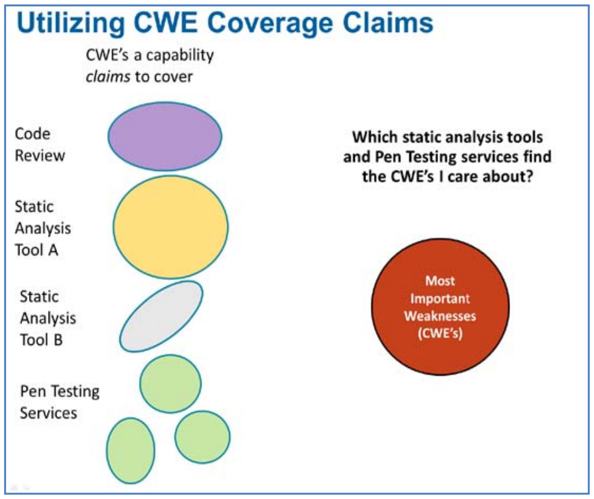
The challenge of gaining assurance about the software is not unique to DoD contracting. Many have attempted to bring the risk from insecure and brittle software under control and these previous efforts have covered some portion of what is now known as Software Assurance. We have reviewed many of these previous works [24, 25, 26, 27, 28. 29. 30, 31, 32], as shown in Figure 11, and believe that many of them created content that could be incorporated within today's Software Assurance efforts as appropriate material for programs to consider when looking for ideas on managing these risks, rather than starting from scratch.

RFP Coverage	[24] System Security Engineering Language for TD Phase RFP	[25] Software Assurance in Acquisition and Contract Language	[26] Guide for Integrating Systems Engineering into DoD Acquisition Contracts s	[27] DoD Open Systems Architecture Contract Guidebook k	[28] Software Assurance in Acquisition: Mitigating Risks to the Enterprise	[29] Recommended Software Assurance Acquisition Language for Space & Missile Systems Center	[30] Cyber Security Language for Information Technology (IT) Requirements	[31] DoD/VA integrated Electronic Health Record (iEHR) Technical Specifications Summary	[32] OWASP Secure Software Contract Annex
C- Description						✓			
J – Top Level Schedule						✓			
J – Prelim. Performance Spec.			✓	✓	✓				✓
J – Program WBS						✓			
J – SOO/SOW	✓	✓		✓	✓	✓	✓	✓	✓
J – SEP			✓			✓			
J – CDRLS			✓			✓			
J – List of Attachments			✓			✓			
L – Instruction to Offerors	✓		✓	✓	✓	✓			
M – Evaluation Factors			✓	✓	✓	✓			

Figure 11: Guidance Coverage Comparisons

3 Use of Automated Detection Software Tools

Identifying most CWEs by hand, or visual inspection, is very difficult, time-consuming, and error-prone. The use of automated tools is needed to effectively find these types of issues in software and your contract should say this explicitly, but must be supplemented with other types of assessment, such as attack surface analysis, a security review of the CONOPs for the application, an architecture review, a design review, fuzz testing, running misuse and abuse test cases, running dynamic analysis tools and web testing tools, conducting pen testing, as well as blue teams and red teams assessment testing of the live application, with the goal is to document the weaknesses targeted by these activities or the attack patterns emulated/simulated/practiced and account for the coverage/completeness of the weakness/attack spaces being addressed [33, 34]. By understanding the coverage of these various types of assessment, as shown in Figure 12, of the different types of assessment with respect to the CWE corpus a mixture of them can be constructed to cover the CWEs of most concern, i.e., the Top-N CWEs that are important to you and the mission the application is supporting, as illustrated in Figure 13.


Utilizing CWE Coverage Claims
CWE's a capability *claims* to cover
Code Review
Static Analysis Tool A
Static Analysis Tool B
Pen Testing Services
Which static analysis tools and Pen Testing services find the CWE's I care about?
Most Important Weaknesses (CWE's)

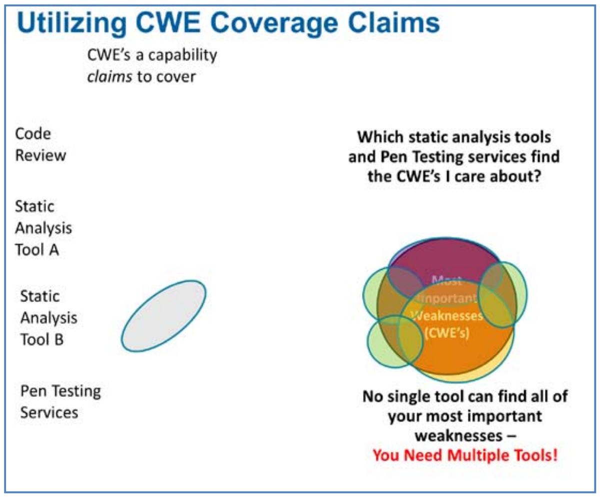

Utilizing CWE Coverage Claims
CWE's a capability *claims* to cover
Code Review
Static Analysis Tool A
Static Analysis Tool B
Pen Testing Services
Which static analysis tools and Pen Testing services find the CWE's I care about?
Most Important Weaknesses (CWE's)
No single tool can find all of your most important weaknesses –
You Need Multiple Tools!

Figure 12: Utilizing CWE Coverage Claims

Figure 13: Covering the Most Important CWEs

Automated software assurance tools must be executed on the entire code base and other assessment approaches may need access to other aspects of the system. Frequently, a contract may only cover an incremental addition to an existing codebase or a subsystem. In these cases the software assurance assessment approaches used should still be applied to the entire codebase and the full system even though the development contract for the new additional section of software code/functionality would only require the repair of defects in the newly written code. The point of assessing all of the system with the software assurance assessment is for the government to gain insight into the software assurance defects in the entire system. This provides the program manager the insight to existing risk within the software release. In a similar vein, the same issues and responsibilities exist for Foreign Military Sales (FMS) code releases as for US Government use software releases.

The DISA STIGS [35] describe in section 5 and specifically 5.4, describe that automatic static analysis should be applied to all of the source code to identify hidden vulnerabilities and weaknesses in the code base.

4 Deliverables: [from the EM-RFP]

[Note: The following table includes a set of CDRLs and the associated Data Item Descriptions (DIDs) that the Program office may include. The selection of the CDRLs can be tailored by the chief engineer based on individual program needs/requirements.]

CDRL #	CDRL Title	Data Item Description (see ASSIST)
SWA001	Technical Report/Study Services (Software Assurance Evaluation Report)	DI-MISC-80508B
SWA002	Technical Report/Study Services (Software Assurance Case)	DI-MISC-80508B
SWA003	Technical Report/Study Services (Vulnerability Assessment Report)	DI-MISC-80508B
SWA004	Security Test Plan	DI-NDTI-81351
SWA005	Technical Report/Study Services (Security Test Report)	DI-MISC-80508B

Table 1: CDRL List

4.1 Attachment 1: Software Assurance Evaluation Report Template [From EM-RFP]

4.1.1 Assessment Report

- Executive Summary
- Objectives and Technical Scope
- Assessment Approach
- Report of Findings
- Vulnerability Descriptions
- Recommendations for Mitigation

OR

4.1.2 Executive Summary

1 Overview
  1.1 Objectives
  1.2 Technical Scope
  1.3 Participants
  1.4 Assessment Approach
2 System Overview
3 Security Analysis

3.1 Critical Findings
3.2 Major Findings
3.3 Minor Findings

4 Recommendations

4.1 Application Risk Assessment Recommendations
4.2 Detailed Risk Assessment (DRA) Recommendations
4.3 Software Vulnerability Assessment (RTA) Recommendations
4.4 Database Security and Compliance Recommendations

Appendix A: Extract of Software Application Tool Scan Results
Appendix B: Extract of Database Tool Scan Results
Appendix C: Security Findings Issue Descriptions
Appendix D: Glossary of Terms

4.2 Software Assurance in CDRL

The CDRL review should occur prior to software acceptance, and at each delivered software set for evaluation:

- Alpha release
- Beta release
- Release Candidate(s)
- Release builds delivered to the government
- Modify list as needed

The CDRL documents which CWEs, CVEs, CAPECs and software assurance critical violations are absent ***and*** present in the delivery-candidate source code. This requires the government to have assessed and defined which CWEs are most important to the project in rank order prior to the CDRL definition. Once this ranked-list of CWEs is created it can be modified throughout the software development lifecycle as needed by the specific program. An example of why you would change the ordered list of CWE might be the discovery of a new critical CWE that affects your program. This new CWE did not exist during the creation of the first Top-N CWE list for your program but is a critical issue that needs to be resolved. This ranked CWE Top-N list must be placed on contract (with appropriately defined Contract Language). The ranked CWE list is sub listed to include a schedule of:

1. CWEs must not be present in the released software
2. CWEs should be removed from the released software
3. CWEs can be removed from the released software
4. All other CWEs are considered acceptable risks by the government team

The above categories can be tied to award fee, or appropriate incentive fee, for the contractor to use in guiding them to produce the highest quality software within the budget and time constraints of the project.

A milestone event should be added to the development milestone list which occurs after CDR and before TRR (and software acceptance). This milestone event would contain the following entrance and exit criteria:

- **Entrance Criteria:** The software developer must submit a report/CDRL to the government entitled "Software Assurance" which defines how the software was developed and verified for software assurance and provides a detailed report of the CWEs, and critical vulnerabilities, present in the delivered software with an adjudication of each CWE, and software assurance critical vulnerability, in the software.
- **Exit Criteria:** The Government validates that all of the CWEs, and software assurance critical violations, as defined by the government on contract are absent from the delivered source code. If the software delivered as described in the report satisfies the government requirements (on contract) then the exit criteria has been met. If the exit criteria has not been met then this milestone remains open and the contractor must repair the software per the government's direction to remove the noted CWEs and software assurance critical violations and both update the CDRL and resubmit the software and report for evaluation.

The Government is responsible for defining which CWEs, in priority order, are to be absent from the delivered software as well as what constitutes a software assurance critical vulnerability.

5 Incentive and award fees

There are several methods being discussed to assist in incentivizing software development contractors to produce higher quality code through software assurance. Each method depends on the contract type being used. For example, one method for an award fee contract is to tie their award fee to the proven absence of your specific Top-N CWEs from your Top-25 CWE list.

X% of the Award Fee for the Contractor is tied to the proven eradication of all items on the Top-25 list

• **Top-1 through Top-5**	**==>**	**0%**	**of X%**
• **Top-1 through Top-10**	**==>**	**20%**	**of X%**
• **Top-1 through Top-15**	**==>**	**40%**	**of X%**
• **Top-1 through Top-20**	**==>**	**60%**	**of X%**
• **Top-1 through Top-25**	**==>**	**100%**	**of X%**

6 Liability

If the contract is written that way, the software developer will only be liable to repair the defects in the software that they, or their subcontractors, have written, when those defects are listed in the government's Top-n CWE list and only to the extent of the software development and testing required to assure the defect has been repaired. The software developer will not be held liable for defects in software, which they, or their subcontractors, did not write or were responsible for. The exception to this rule applies to open source software for which the software developer, or their subcontractors, made the decision to include into the software product. In this case the software developer, or their subcontractors, shall be liable to repair only the defects in the open source code software that are listed in the government's Top-n CWE list.

7 References

[1] National Defense Authorization Act for Fiscal 2013, Public Law 112–239, Section 933, Jan. 2013 http://www.gpo.gov/fdsys/pkg/PLAW-112publ239/pdf/PLAW-112publ239.pdf

[2] USD(AT&L) Memorandum, Document Streamlining – Program Protection Plan (PPP), July 18, 2011. http://www.acq.osd.mil/se/docs/PDUSD-ATLMemo-Expected-Bus-Practice-PPP-18Jul11.pdf

[3] Program Protection Plan Outline and Guidance version 1.0, July 2011. http://www.acq.osd.mil/se/docs/PPP-Outline-and-Guidance-v1-July2011.pdf

[4] Defense Acquisition Guidebook Chapter 13, Program Protection https://acc.dau.mil/dag13

[5] Common Attack Pattern Enumeration and Classification (CAPEC™) - a publicly available, community-developed list of common attack patterns along with a comprehensive schema and classification taxonomy - to analyze environments, code, and interfaces for common destructive attack patterns, http://capec.mitre.org/. CAPEC is a catalog of attack patterns along with a comprehensive schema and classification taxonomy focused on enhancing security throughout the software development lifecycle, and to support the needs of developers, testers and educators. By providing a standard mechanism for identifying, collecting, refining, and sharing attack patterns among the software community, CAPEC provides for a more complete and thorough review of the strength of our systems from the point-of-view of attackers.

[6] ITU-T Telecommunication Standardization Sector of ITU, X.1544 Series X: Data Networks, Open System Communications and Security, Cybersecurity information exchange – Event/incident/heuristics exchange, Common attack pattern enumeration and classification, Apr. 2014 http://www.itu.int/rec/T-REC-X.1544-201304-I

[7] Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today, Feb. 2011 http://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf

[8] Practical Security Stories and Security Tasks for Agile Development Environments, Jul. 2012 http://safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf

[9] ISO/IEC TR 20004-1, Information technology — Security techniques — Refining Software vulnerability analysis under ISO/IEC 15408 and ISO/IEC 18045 — Part 1: Using publicly available information security resources, 2012 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50951

[10] ISO/IEC TR 20004-2, Information technology — Security techniques — Refining software vulnerability analysis under ISO/IEC 15408 and ISO/IEC 18045 — Part 2: CWE and CAPEC based software penetration testing.

[11] NIST SP800-51 revision 1, Guide to Using Vulnerability Naming Schemes, Feb 2011 (http://csrc.nist.gov/publications/nistpubs/800-51-rev1/SP800-51rev1.pdf

[12] NIST SP800-53 revision 4, Security and Privacy Controls for Federal Information Systems and Organizations, Apr 2013 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

[13] Office of the Deputy Assistant Secretary of Defense (ODASD) Systems Engineering (SE) Trusted Systems and Networks (TSN) Analysis, June 2014 http://www.acq.osd.mil/se/docs/Trusted-Systems-and-Networks-TSN-Analysis.pdf

[14] Office of the Deputy Assistant Secretary of Defense (ODASD) Systems Engineering (SE) Software Assurance Countermeasures in Program Protection Planning, March 2014 http://www.acq.osd.mil/se/docs/SwA-CM-in-PPP.pdf

[15] Defense Acquisition Guidebook Chapter 13, Program Protection, May 2013 https://acc.dau.mil/docs/dag_pdf/dag_ch13.pdf

[16] Institute for Defense Analyses (IDA) State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation, Jul 2014 (http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf)

[17] Common Weakness Enumeration (CWE™) - A Community-Developed Dictionary of Software Weakness Types - to examine software architectures, designs, and source code for weaknesses. http://cwe.mitre.org. Targeted to developers and security practitioners, CWE) is a formal or dictionary of common software weaknesses created to serve as a common language for describing software security weaknesses in architecture, design, or code; serve as a standard measuring stick for software security tools targeting these weaknesses, and to provide a common baseline standard for weakness identification, mitigation, and prevention efforts.

[18] ITU-T Telecommunication Standardization Sector of ITU, X.1524 Series X: Data Networks, Open System Communications and Security, Cybersecurity information exchange – Event/incident/heuristics exchange, Common weakness enumeration, Mar. 2012 http://www.itu.int/rec/T-REC-X.1524-201203-I/

[19] CWE/SANS Top 25 Most Dangerous Software Errors - http://cwe.mitre.org/top25/. The Top 25 is a consensus list of the most significant software errors that can lead to serious software vulnerabilities. The errors are dangerous because they frequently will allow attackers to completely take over the software, steal data, or prevent the software from working at all. The Top 25 is the result of collaboration between the SANS Institute,

MITRE, and many top software security experts in the US and Europe and leverages experiences in the development of the SANS Top 20 attack vectors and MITRE's CWE.

[20] Common Vulnerabilities and Exposures (CVE®) - The Standard for Information Security Vulnerability Names. http://cve.mitre.org/. International in scope and free for public use, CVE is a dictionary of publicly known information security vulnerabilities and exposures. CVE's common identifiers enable data exchange between security products and provide a baseline index point for evaluating coverage of tools and services.

[21] ITU-T Telecommunication Standardization Sector of ITU, X.1520 Series X: Data Networks, Open System Communications and Security, Cybersecurity information exchange – Event/incident/heuristics exchange, Common vulnerabilities and exposures, Jan. 2014 http://www.itu.int/rec/T-REC-X.1520-201401-I

[22] Software Product Liability: Understanding and Minimizing the Risks and Michael D. Scott, Tort Liability for Vendors of Insecure Software: Has the Time Finally Come?, Berkeley Technology Law Journal (BTLJ) Volume 5, 67 Md. L. Rev. 425, 2008. http://btlj.org/1990/05/26/volume-5-issue-1-spring-1990/

[23] AFLCMC/EZC – Engineering Model RFP Language, Hanscom Air Force Base, MA, November 2012.

[24] "System Security Engineering Language for TD Phase RFP" - Deputy Assistant Secretary of Defense for Systems Engineering (DASD(SE)). 2014. "Suggested Language to Incorporate System Security Engineering for Trusted Systems and Networks into Department of Defense Requests for Proposals." Washington, D.C.: DASD (SE). http://www.acq.osd.mil/se/docs/SSE-Language-for-TSN-in-DoD-RFPs.pdf

[25] Software Assurance in Acquisition and Contract Language Acquisition and Outsourcing, Volume I, Version 1.2, May 18, 2012 https://buildsecurityin.us-cert.gov/swa/software-assurance-pocket-guide-series#acquisition

[26] Guide for Integrating Systems Engineering into DoD Acquisition Contracts, version 1.0, December 11, 2006 http://www.acq.osd.mil/se/docs/Integrating-SE-Acquisition-Contracts_guide_121106.pdf

[27] DoD Open Systems Architecture Contract Guidebook, version 1.1, June 2013 https://acc.dau.mil/osaguidebook

[28] Software Assurance in Acquisition: Mitigating Risks to the Enterprise, National Defense University Press. Polydys, M. & Wisseman, S., 2009. http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA495389

[29] Recommended Software Assurance Acquisition Language for Space & Missile Systems Center, HQ Space and Missile Systems Center, SMC/ENP, September 2015.

[30] USTRANSCOM - Information Assurance/Cyberspace Operations Defense Functional Requirements, Cyber Security Language for Information Technology (IT) Requirements http://www.transcom.mil/about/org/tccs/Cyber_Defense_IT_Contract_Language.pdf

[31] DoD/VA integrated Electronic Health Record (iEHR) Technical Specifications Summary, sections 6.5.1 through 6.5.7, Version 2.2, June 17, 2013.

[32] Open Web Application Security Project, OWASP Secure Software Contract Annex, https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex

[33] Deputy Assistant Secretary of Defense for Systems Engineering and Department of Defense Chief Information Officer, Software Assurance Countermeasures in Program Protection Planning, March 2014. http://www.acq.osd.mil/se/docs/SwA-CM-in-PPP.pdf

[34] Integrated Analysis and Reporting In Multiple Tools - What cybersecurity and robustness testing tool manufactures should be building towards, 2015. https://interact.gsa.gov/sites/default/files/Mon%20AM2-Integrated%20Analysis%20and%20Reporting%20In%20Multiple%20Tools%20V%208-30-2015_MVE%20-RAM.pdf

[35] DISA Application Security and Development Security Technical Implementation Guide (STIG), Version 3, Rel.10", 23 January 2015. http://iase.disa.mil/stigs/app-security/app-security/Pages/index.aspx