Software Assurance Countermeasures in Program Protection Planning

MARCH 2014

Deputy Assistant Secretary of Defense for Systems Engineering and Department of Defense Chief Information Officer

Washington, D.C.

Deputy Assistant Secretary of Defense for Systems Engineering (DASD(SE)) and Department of Defense Chief Information Officer (DoD CIO). 2014. *Software Assurance Countermeasures in Program Protection Planning.* Washington, D.C.: DASD(SE) and DoD CIO.

Office of Primary Responsibility:

Deputy Assistant Secretary of Defense
Systems Engineering
3030 Defense Pentagon
3C167
Washington, D.C. 20301-3030
www.acq.osd.mil/se

Contents

Introduction

Department of Defense (DoD) systems incorporate an extensive amount of software, and therefore defense programs must conduct early planning to impose software assurance countermeasures to counter adversarial threats that may target that software. Programs must ensure systems are securely supplied, designed, and tested to ensure mission success and to protect critical functions, associated components, and critical program information (CPI). Of particular interest are protection and assurance activities undertaken during the integration and development of commercial off-the-shelf (COTS) components; activities designed to mitigate attacks against the operational system (the fielded system); and activities that address threats to the development environment.

The purpose of the software assurance countermeasures section of the Program Protection Plan (PPP) is to help programs develop a plan and statement of requirements for software assurance early in the acquisition lifecycle and to incorporate the requirements into the request for proposal (RFP). Programs then use that plan to track software assurance protections throughout the acquisition. The progress toward achieving the plan is measured by actual accomplishments/results that are reported at each of the Systems Engineering Technical Reviews (SETR) and recorded as part of the PPP.

The *Program Protection Plan (PPP) Outline and Guidance* (2011) requires acquisition programs to address software assurance responsibilities for planning and implementing program protection countermeasures. Such countermeasures address the anticipated attacks a system may experience by eliminating or reducing vulnerabilities. The countermeasures are selected with an understanding of which parts of the software are the most critical to the success of the mission. The plan includes a sample Software Assurance Countermeasures table (figure 1), which summarizes the planned and current state of a program's software assurance activities. The table is also used as part of a vulnerability assessment to identify operational, developmental, design, COTS, and software tool vulnerabilities that can be addressed by planning and implementing software assurance countermeasures.

The table in the PPP is divided into three sections that provide different vulnerability and countermeasure perspectives on software assurance plans and implementation:

Development Process – assurance activities conducted during the development process to mitigate and minimize attacks (e.g., threat assessment and modeling, attack surface analysis, architecture and design reviews, application of static and dynamic code assessment tools and services, penetration testing, and red teaming) that the developed system is likely to face when deployed into operation

Operational System – attack countermeasures and other assurance activities applied within the operational environment (e.g., failover, fault isolation, encryption, application firewalls, least privilege, and secure exception handling) to mitigate attacks against the delivered system and software interfaces, which may include COTS, Government off-the-shelf (GOTS), open source, and other off-the-shelf software

Development Environment – assurance activities and controls (e.g., access controls, configuration management, and release testing) applied to tools and activities (e.g.,

Development Process								
Software (CPI, critical function components, other software)	**Static Analysis p/a (%)**	**Design Inspect**	**Code Inspect p/a (%)**	**CVE p/a (%)**	**CAPEC p/a (%)**	**CWE p/a (%)**	**Pen Test**	**Test Coverage p/a (%)**
Developmental CPI SW	100/80	Two Levels	100/80	100/60	100/60	100/60	Yes	75/50
Developmental Critical Function SW	100/80	Two Levels	100/80	100/70	100/70	100/70	Yes	75/50
Other Developmental SW	none	One level	100/65	10/0	10/0	10/0	No	50/25
COTS CPI and Critical Function SW	Vendor SwA	Vendor SwA	Vendor SwA	0	0	0	Yes	UNK
COTS (other than CPI and Critical Function) and NDI SW	No	No	No	0	0	0	No	UNK

Operational System						
	Failover Multiple Supplier Redundancy (%)	**Fault Isolation**	**Least Privilege**	**System Element Isolation**	**Input Checking / Validation**	**SW Load Key**
Developmental CPI SW	30	All	all	yes	All	All
Developmental Critical Function SW	50	All	All	yes	All	all
Other Developmental SW	none	Partial	none	None	all	all
COTS (CPI and CF) and NDI SW	none	Partial	All	None	Wrappers/ all	all

Development Environment								
SW Product	**Source**	**Release Testing**	**Generated Code Inspection p/a (%)**					
C Compiler	No	Yes	50/20					
Runtime libraries	Yes	Yes	70/none					
Automated test system	No	Yes	50/none					
Configuration management system	No	Yes	NA					
Database	No	Yes	50/none					
Development Environment Access	Controlled access; Cleared personnel only							

FIGURE 1 – SOFTWARE ASSURANCE COUNTERMEASURES (SAMPLE)

compilers, linkers, integrated development environments, run-time libraries, and test harnesses) used to develop and sustain software to mitigate attacks

Given the constraints of cost, schedule, and performance, fully comprehensive assessment and testing often are not feasible. Thus software assurance planning should reflect priorities chosen to mitigate risk and deliver mission capability with acceptable levels of assurance. The coding language, source of code (i.e., custom, COTS, GOTS, open source), platform (i.e., web-based, mobile, embedded, etc.) as well as the results of criticality analysis (see Defense Acquisition Guidebook (DAG) 13.3.2.1) will be used to prioritize software assurance activities when planning for software assurance.

Development Process

The purpose of this section of the table is to identify, set goals for, and track the assurance activities conducted during software development and the integration of off-the-shelf components. As appropriate to the risk of compromise and criticality of the software in question, program managers (PM) are to analyze the development activities for:

- Potential introduction of vulnerabilities and risks based on the anticipated threat and the attacks the threats are capable of making against the system;
- Development of a plan for the assurance process as well as the technical disciplines and knowledge needed for Integrated Project Teams (IPT);
- How IPTs address the architecture, design, code, and implementation choices to include the appropriate mitigations necessary to address the anticipated attacks and assure the critical function software components; and
- Review points to track/assess the progress at the milestones in the PPP.

Not all software will require the same level of software assurance activities and mitigation planning and implementation. In programs with millions of lines of code, there may be some functions (perhaps a monthly reporting feature) that are less mission-critical than others (perhaps a satellite station-keeping module). It may also be difficult to perform some types of assessment and mitigation activities on COTS software for which the source code is not available. Note that in such cases software-related risks still exist and may be unmitigated. The software assurance table in the PPP recognizes these varying types of software and allows for differing plans/implementation of assurance as needed.

Static Analysis

Programs should investigate the applicability of automated static analysis tools to review source and/or binary copies of their software and, where advantageous, apply both static source code and static binary analysis to assist in identifying latent weaknesses that would manifest as operational system vulnerabilities and allow attackers to interfere, manipulate, or otherwise suborn the system's mission capabilities. The use of these types of tools within the development activity (i.e., as an add-on to the developer's Integrated Development Environment (IDE)) as well as in the Independent Test and Evaluation (IT&E) activities is both valuable and useful. Approaches that integrate such forms of continuous assessment into the developer's activities should be emphasized and encouraged.

Design Inspection

The establishment and update of secure design and code standards by the program should address the potential types of attacks the system would face and draw upon DoD, Government, Federally Funded Research and Development Centers (FFRDC), academia, commercial websites, and industry sources for mitigation approaches and methods to address those that could affect the system's mission capabilities. The list of attack patterns captured in the Common Attack Pattern Enumeration and Classification (CAPEC™) collection can be used to help consistently analyze a system for potential types of attacks the system may face. Lists such as CAPEC can also bring consistency into the process of verifying that the design and coding standards are being followed.

Code Inspection

Because of the subtle nature of most weaknesses in code that lead to unreliable, insecure, and brittle applications that are easily influenced by attackers, it is important that code inspections using appropriate tools be part of the approach used to minimize these weaknesses. The Common Weakness Enumeration (CWE) catalog captures more than 700 types of weaknesses in code, design, architecture, and implementation, but not all of them are equal threats to any specific application or system. Programs may wish to draw upon secure design and coding approaches defined on websites such as Top 10 Secure Coding Practices"[1] and the Common Weakness Enumeration (CWE)/ SysAdmin, Audit, Network, Security (SANS) Top 25 Most Dangerous Software Errors[2] to establish and update their secure design and coding standards. As a minimum, the code inspection is used to inspect for conformance to the secure design and coding standards established for the program.

An important part of the code inspection is to identify the subset of the overall CWE collection to focus on initially. Alternate approaches to focusing in on a subset of the weaknesses are described in the CWE and CAPEC sections that follow. These approaches can be used independently or in combination if desired.

Because of the dynamic nature of the threat environment and information about how systems can be compromised through software weaknesses, the program should have a methodology to periodically update the secure design and coding standards so that reviews using the standards address new types of attacks and types of weaknesses.

The next three sections of this document describe the middle three columns of the PPP Software Assurance Table, which are meant to capture how the established vulnerability (CVE), weakness (CWE), and attack pattern (CAPEC) collections are being used by the project team to identify and mitigate the most dangerous types of vulnerabilities in the software. These columns are further defined below.

Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures (CVE) information is used to identify, track, and coordinate mitigation activities of the publicly known vulnerabilities in commercial (COTS) and open source software that are often used by malicious actors/agents to attack systems. Programs that incorporate COTS software into their systems should perform regular searches of the CVE lists before purchase and throughout the software lifecycle to understand vulnerabilities in those COTS software components and assess potential threats to mission success.

[1] https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+...
[2] http://cwe.mitre.org/top25/index.html

The CVE list is a compilation of publicly known information about security vulnerabilities and exposures. The list is international in scope, free for public use, and referenced in most commercial tools that scan operational systems and networks for vulnerabilities. The CVE list can be used to identify publicly known software vulnerabilities that could:

- Allow an attacker to execute unauthorized code or commands;
- Allow an attacker to gain privileges or assume identities;
- Allow an attacker to access and/or manipulate data in a way that is contrary to the specified access restrictions for that data;
- Bypass protection mechanisms;
- Allow an attacker to hide their activities; and
- Allow an attacker to conduct denial of service attacks.

CVE is intended for use by security experts, so it assumes a certain level of knowledge. Programs should use a tool during incremental software testing of their commercial and open source packages to scan operational components and match the results with the CVE dictionary. Alternatively, a program can review the CVE list for any publicly known vulnerabilities for the software packages being used by that program. A list of CVE-compatible tools is available at http://cve.mitre.org/compatible/product.html.

The CVE column in the PPP Software Assurance Countermeasures table reports the planned and actual (p/a) percentages of software components that incorporate COTS or open source that have been analyzed and acceptably remediated against the CVEs from the CVE list that apply to those COTS and open source packages.

Supportive analysis by the project team must record the CVEs found, the remediation applied, and the residual risk to the mission of any unresolved CVEs. To identify which CVEs should be included in the analysis, the list of CVEs for each COTS product and open source should be tracked and those that were remediated documented as such. For each COTS and open source package used as part of the system, the project staff should determine whether an explicit vulnerability advisory/alert activity is provided/offered by the provider/developer of those packages.

For those software developers that do not provide publicly available advisories/alerts about security issues that need to be resolved, the project staff should carefully consider the risk they are inheriting from that developer. Without CVE identifiers it is much harder to track and manage the state of deployed software within the DoD's vulnerability management practice and the automation tooling deployed within the DoD. All developmental CPI software and developmental critical-function software packages, whether COTS or open source, must be evaluated using CVE, to reveal exposures inherited by incorporating open source or COTS libraries or products.

Guidance on searching the CVE is located at http://cve.mitre.org/about/faqs.html#c. An important aspect of applying CVE tools and reviews to a collection of COTS and open source is to apply the Common Vulnerability Scoring System (CVSS) to the determination of which CVEs to mitigate first and to understand the severity of the remaining CVEs.

If the selected tool identifies any CVE with a CVSS score above medium (4), programs should mitigate the vulnerability with highest priority first and then work through the next highest priority issue until the residual risk represented by the remaining vulnerabilities is acceptable to the mission owner. CVEs that are included in any DoD Information Assurance Vulnerability Management (IAVM) alerts and advisories should be addressed in accordance with the priorities and timeframe included in the IAVM from the Defense Information Systems Agency (DISA).

The CVE website is at http://cve.mitre.org

Common Attack Pattern Enumeration and Classification (CAPEC)

Common Attack Pattern Enumeration and Classification (CAPEC) is meant to be used for the analysis of common patterns of attacks against systems, whether for understanding how attacks are committed, scoping of relevant threats, templates for malicious testing, or as a foil for thinking about the susceptibility of system's architecture, design, and technical implementation to specific attacks.

CAPEC is an international, free catalog of attack patterns outlining information such as a comprehensive description of the phases and steps in attacks, the weaknesses they are effective against (using CWEs), and a classification taxonomy that can be used for the analysis of common attack patterns. CAPEC attack patterns cover a wide variety of families of attacks, including: data leakage attacks, resource depletion attacks, injection attacks, spoofing attacks, time and state attacks, abuse of functionality attacks, attacks using probabilistic techniques, attacks exploiting authentication, attacks exploiting privilege/trust, attacks exploiting data structure, resource manipulation attacks, network reconnaissance, social engineering attacks, and some physical security attacks and supply chain attacks.

The attack patterns in CAPEC can be a powerful mechanism to capture and communicate the attacker's perspective, organize the analysis of a system with respect to attacks, and prioritize weaknesses (CWEs) based on the anticipated attack patterns. They are descriptions of common methods for exploiting software. Identified attack patterns may influence the selection of the COTS and open source software products, programming languages, and design alternatives. By understanding the attacker's perspective and how a program's software is likely to be attacked, programs can directly consider these exploit attempt methods and mitigate them with design, architecture, coding, and deployment choices that will lead to more secure software.

Programs should identify the set of attack patterns that pose the most significant risk and should leverage them at each stage of the Software Development Lifecycle (SDLC). A discussion of how to use CAPEC in this manner is available on the "Engineering for Attack" page on the CWE site (http://cwe.mitre.org/community/swa/attacks.html). This is the same basic methodology described in the new ISO/IEC Technical Report 20004, "Refining Software Vulnerability Analysis under ISO/IEC 15408 and ISO/IEC 18045, which describes an alternate approach for doing a vulnerability analysis of a software-based system under the Common Criteria regime. ISO/IEC 15408 and ISO/IEC 18045 are the two standards that guide and describe the Common Criteria evaluation methodology.

The Engineering for Attack page describes how to use attack patterns to identify those attacks and weaknesses that are of most concern. Such a list can drive better choices in design, architecture, planned operational use, security policies, requirements, and generally thinking through the risks related to the system's intended use. This list can identify a manageable set of relevant CWE weaknesses to avoid in design/coding and to inspect against during implementation and verification. The list's associated CAPECs can inform test and evaluation by identifying high-priority test cases for risk-based security testing, penetration testing, and red teaming.[3]

Supportive analysis by the project team should record

- the CAPECs identified as germane to the system,
- the CWEs identified as being susceptible to those CAPECs and
- the remediation applied along with
- an understanding of the residual risk to the mission of any CWEs that were not tested by simulating CAPECs against the system.

To identify which CWEs should be included in the testing analysis, the list of CWEs should be tracked and those that can be covered by the application of an available analysis tool/service appropriately remediated. For each CWE not covered by an available static analysis tool/service, the project staff should determine whether an appropriate CAPEC-inspired test case or Red Team activity was conducted without finding an exploitable CWE.

For those CWEs that were not covered by static analysis or testing, the project staff should carefully consider the risk to the mission from the potential of those weaknesses remaining in the system.

The CAPEC website is http://capec.mitre.org. A description of the CAPEC schema is located in the "Documentation" portion of the CAPEC Documents page at http://capec.mitre.org/about/documents.html.

Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is international in scope and free for public use. CWE provides a unified, measurable set of software weaknesses to enable more effective discussion, description, selection, and use of software security tools and services to find

[3] http://capec.mitre.org/documents/An_Introduction_to_Attack_Patterns_as_a_Software_Assurance_Knowledge_Resource.pdf

weaknesses in source code and operational system components as well as to better understand and manage software weaknesses related to architecture and design.

CWE is targeted to developers and security practitioners. Programs should use CWE-compatible tools to scan software for CWE. A list of CWE-compatible products is available at http://cwe.mitre.org/compatible/product.html.

The CWE column in the table reports the planned and actual percentages of developed software components that have been evaluated utilizing the weaknesses from the CWE list to identify the appropriate subset of CWEs, to consider alternate design and architectures or alternate coding constructs.

The CWE/SANS Top 25 Most Dangerous Software Errors list on the CWE and SANS websites provides detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them.

The Common Weakness Risk Analysis Framework (CWRAF) methodology is described on the CWE website and numerous examples are provided to help a project team learn how to apply the methodology to their system in combination with the Common Weakness Scoring System (CWSS).

By using the Common CWSS, a program also can reflect its specific list of dangerous CWEs into its tools so the risk to the mission of the weaknesses found during static and dynamic analysis or penetration testing reflects the relative importance of those impacts.

The CWE website is at http://cwe.mitre.org, and the CWSS web page is at http://cwe.mitre.org/cwss/.

In addition, the project team should have a documented understanding of the residual risk to the mission of any CWEs that were not the subject of review by static analysis tools/services or tested by simulating the CAPECs that would be effective against those CWEs. For CWEs deemed to be dangerous but not covered by a static analysis tool/service, the project staff should determine whether an appropriate CAPEC-inspired test case or Red Team activity was conducted without finding an exploitable CWE.

For those CWEs that were not covered by static analysis or testing, the project staff should carefully consider the risk to the mission from the potential of those weaknesses remaining in the system. Without demonstrable evidence that the CWEs that an attacker could exploit are mitigated, there will always be some level of risk, but it is incumbent on the project staff to document this residual risk for the end user so the user can manage that risk when the system is deployed within the DoD. All developmental CPI software and developmental critical-function software should be evaluated against the identified subset of the CWE list.

In addition to the above-listed MITRE websites, PMs should consider best practices identified at http://www.safecode.org/index.php.

Penetration Test

Programs should report what portion of the system will undergo penetration testing. The purpose of penetration testing is to subject the system to an attack exercise to raise awareness of exploitable vulnerabilities in the system and accelerate their remediation. Also the knowledge that a system will undergo penetration testing increases the vigilance of the software engineers responsible for architecting, designing, implementing, and fielding the systems.

The text should support the number with a brief explanation of the penetration testing performed and a reference to any supporting reports generated by that testing.

The units used for planned/actual percentages for this metric are at the discretion of the program. They should be explained in the text and should be meaningful and provide insight into the completeness of the testing. For example, a network that exposes a certain number of protocols may measure the percentages in the space of protocol states. A system with an application programming interface (API) may measure the number of interface functions probed.

Test Coverage

Programs should report on their planned and actual test coverage. Units and metrics for test coverage are at the discretion of the program but should be meaningful and yield insight into the completeness of the testing regimen.

Possible measures for test coverage include percentage of statements exercised, percentages of API calls and exception conditions exercised, or number of function points tested.

Operational System

This section refers to the software and firmware on the fielded system. Software assurance countermeasures is a rapidly evolving area. Successful assessments, techniques, applications, and example outcomes are frequently published in papers that can be found at DoD, Government, FFRDC, and commercial websites. The FFRDC Carnegie Mellon Software Engineering Institute (SEI) and MITRE both have searchable libraries containing information about the approaches to Software Assurance indicated in the *Program Protection Plan Outline and Guidance*, Table 5.3.3-1 Application of Software Assurance Countermeasures.

Failover Multiple Supplier Redundancy

Identical code for a failed function will most likely suffer the same failure as the original. For redundancy in software, therefore, a completely separate implementation of the function is needed. This independence reduces the probability that the failover code will be susceptible to the same problem.

Fault Isolation

Software mechanisms that isolate faults include functions to trap, log, and otherwise protect element failures from affecting other elements and the larger system. Logs help trace the sources of operational faults. Logs also can be examined to help assess whether the fault is indicative of a malicious attack.

Programs reporting a "Yes" in the table should be prepared to elaborate with technical detail on how the fault isolation mechanisms were employed in the architecture and design for the particular component or subsystem.

Least Privilege

The principle of least privilege dictates that one should limit the number, size, and privileges of system elements. Least privilege includes separate user roles, authentication, and limited access to enable all necessary functions but minimize adverse consequences of inappropriate actions. Thus should a system element fall under the control of an attacker, the actions that attacker can take may be constrained.

Programs reporting a "Yes" in the table should be prepared to elaborate with technical detail on how least privilege principles were employed in the architecture and design for the particular component or subsystem.

System Element Isolation

Software following the principle of system element isolation allows system element functions to operate without interference from other elements. Such isolation limits the cascading effect that could ensue due to compromise of a single element.

Programs reporting a “Yes” in the table should be prepared to elaborate with technical detail on how system element isolation principles were employed in the architecture and design for the particular component or subsystem.

Input Checking/Validation

Input checking and validation should ensure that out-of-bounds values and out-of-sequence operations are handled without causing failures and that the invalid input events are logged. This checking may be applied to developmental software through coding guidelines and review. It may also apply to COTS and Non-Developmental Item (NDI) software through constructs such as wrappers and input filtering.

Programs reporting a “Yes” in the table should be prepared to elaborate on the architectural and design criteria governing the extent of input checking/validation employed.

Software Load Key

Software load key refers to mechanisms by which executable software code is encrypted or otherwise protected (e.g., cryptographic checksums, digital signatures, secure boot) from corruption, between factory delivery and use in a military mission.

Programs reporting a “Yes” in the table should be prepared to elaborate on specific techniques that are included in the architecture, design, and implementation of the software component or subsystem to guarantee the integrity of the software image and detect any unauthorized modification of the software once deployed.

Development Environment

Software tools used in the development environment (as opposed to the actual fielded software) are another source of risk to warfighting capability and should be considered in the PPP. In particular, an attacker could use a compromised development environment to insert malicious code, exploitable vulnerabilities, and/or software backdoors into the operational software before it is fielded.

Examples of software development tools include:

- Compilers, assemblers, pre-compilers, and other code-generating tools such as design templates
- Structured code editors
- Code static analysis tools
- Debugging and timing analysis tools
- Code configuration management tools
- Accounts and access controls on development computers and networks
- Test management tools, test data generators, test harnesses, automated regression testing tools

Examples of compromising tools to achieve malicious insertion include

- Modify compiler to generate or insert additional functionality into the operational code
- Modify a math library of routines with malware that then will be incorporated into the operational code.

Programs should tailor the list contents of the SW Product column in this section of the table to enumerate the software tools pertinent to the program's development environment(s). For each SW product listed, table entries should address the items enumerated in the following columns.

Source Code Availability

When source code is available, it becomes easier to answer some questions about the behavior of the tool and to detect potential compromise.

Is source code available for the tool? A yes or no response in this column may suffice. If further information (e.g., coding language, code size, licensing cost constraints) would provide useful insight, annotate the entry with a note.

Release Testing

Software tools are often updated. These updates are a potential path for an attacker to compromise the development environment and thus the operational software.

Indicate whether testing for indications of malicious insertion or tool compromise are performed on each update of the tool before that update is incorporated into the development environment.

Generated Code Inspection

Indicate whether/how any generated code for the system is examined for malicious code or exploitable vulnerability potentially inserted by the software tool in question.

In general, the problem of how to effectively inspect generated code for malicious insertion remains an open area of research. From the practical standpoint, it is better to perform some inspection than to ignore the problem entirely. That inspection at least raises the bar for what an attacker needs to do to compromise the system undetected.

Potential code inspection countermeasures include:

- Manual inspection of a representative sample of the generated code
- Analysis of the code with reverse engineering tools
- Identification of the libraries compiled into an executable
- Comparison to baselines generated by previous versions of the tool
- Manual inspection of tool outputs against a known/analyzable test corpus
- Advanced/experimental techniques such as automated function extraction

Note that in many instances simple checks can be effective in detecting some injected malware. For example: extracting, comparing, and sorting strings might point to a trigger string used to open a backdoor. Decompiling an executable may reveal the presence of operation codes not normally generated by the compiler.

Where generated code inspection is deemed of benefit, programs should tailor the inspection to the unique aspects of the program and report planned and actual percentages appropriately.

Additional Countermeasures

Programs should consider adding columns to this area of the software assurance table with the rationale for the additions if programs judge them to significantly reduce the risk of malicious insertion. Additional countermeasures may include:

- Access controls and other controls detect malicious behavior or suspicious artifacts in the development environment.
- Information assurance controls to safeguard technical data in the development environment (networks, computers, test equipment, and configuration systems).
- Controlling and accounting for printing of technical manuals and other documentation.

Acronyms

API	application programming interface
CAPEC	Common Attack Pattern Enumeration and Classification
CF	Critical Function
CIO	Chief Information Officer
COTS	commercial off-the-shelf
CPI	critical program information
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
CWSS	Common Weakness Scoring System
DAG	Defense Acquisition Guidebook
DASD(SE)	Deputy Assistant Secretary of Defense for Systems Engineering
DISA	Defense Information Systems Agency
DoD	Department of Defense
FFRDC	Federally Funded Research and Development Center
GOTS	Government off-the-shelf
IAVM	Information Assurance Vulnerability Management
IDE	Integrated Development Environment
IPT	Integrated Product Team
IT&E	Independent Test and Evaluation
NDI	Non-Developmental Item
p/a	planned/actual
pen test	penetration test
PM	program manager
PPP	Program Protection Plan
RFP	request for proposal
SANS	SysAdmin, Audit, Network, Security
SDLC	Software Development Lifecycle
SEI	Software Engineering Institute
SETR	Systems Engineering Technical Review
SW	software
SwA	software assurance
UNK	unknown

References

Defense Acquisition Guidebook. Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics. https://dag.dau.mil/.

Department of Defense Instruction (DoDI) Interim 5000.02. 2013. “Operation of the Defense Acquisition System.” Under Secretary of Defense for Acquisition, Technology, and Logistics (November 25). http://www.dtic.mil/whs/directives/corres/pdf/500002_interim.pdf

Program Protection Plan Outline and Guidance, Version 1.0. 2011. Washington, D.C.: Deputy Assistant Secretary of Defense for Systems Engineering. http://www.acq.osd.mil/se/docs/PPP-Outline-and-Guidance-v1-July2011.pdf