Streamlining Information Assurance Testing With Automation

Christopher Huey, Charles McElveen
Cobham Analytic Solutions
Orlando, FL
Chris.Huey@cobham.com,
Charles.McElveen@cobham.com

Kelly Djahandari
Northrop Grumman
Orlando, FL
Kelly.Djahandari@ngc.com

ABSTRACT

System complexity, aggressive schedules, and limited resources are risks a program must overcome in order to properly implement and maintain system security configuration controls during development, integration, fielding, and operations. Vulnerabilities caused by inadequate system security configurations create opportunities for adversaries to successfully conduct cyber attacks on systems. A driving factor contributing to this challenge is the lack of efficient methods for verifying the system security configurations comply with the security requirements. Identifying and reporting vulnerabilities in a timely manner are critical for effectively mitigating identified risks. Automated test tools exist for assisting in the process, but many conduct generic test inspections and are not tailored to verify the specific security policies and requirements established for the system.

This paper describes a process used by the Combat Air Force (CAF) Distributed Mission Operations Network (DMON) Cross Domain Solution team to effectively identify and mitigate security vulnerabilities during system development, integration, and deployment. The process leveraged automated tools and an associated strategy to streamline the Information Assurance testing effort and increase the cyber security posture of the DMON Cross Domain Solution. The paper addresses process enhancements implemented to establish and sustain a high level of security assurance required in the warfighter's integrated live, virtual, constructive training environments.

ABOUT THE AUTHORS

Christopher Huey, CISSP, is a Principle Cyber Security Engineer with over 24 years experience in information assurance on Department of Defense (DoD) and Intelligence Community programs. He is currently supporting the Distributed Mission Operations Network Cross Domain Solution Services program by supporting security testing, certification and accreditation activities, and process improvement tasks. Mr. Huey received his Bachelor's Degree in Computer Science from the University of Michigan and his Master's Degree in Systems Engineering from George Mason University.

Charles McElveen, CISSP, ISSEP, is a Principle Cyber Security Engineer with over 27 years of progressive experience supporting a variety of military and DoD programs. He is currently supporting the Distributed Mission Operations Network Cross Domain Solution Services tasking. He received his Bachelor's Degree in Computer Science from the University of Southern Mississippi and a Master's Degree in Management/Management Information Systems from Florida Institute of Technology.

Kelly Djahandari, CISSP, is a Cyber Architect and is leading the Cross Domain Solution Research and Development task order under the Distributed Mission Operations Mission Training program. Her information assurance experience includes more than 16 years of software engineering in network security research and cross domain solutions. She received a Bachelor's Degree from George Mason University and a Master's Degree from the University of Virginia.

Streamlining Information Assurance Testing With Automation

Christopher Huey, Charles McElveen
Cobham Analytic Solutions
Orlando, FL
Chris.Huey@cobham.com,
Charles.McElveen@cobham.com

Kelly Djahandari
Northrop Grumman
Orlando, FL
Kelly.Djahandari@ngc.com

INTRODUCTION

Many programs are faced with limited budgets and compressed schedules, so the need to streamline processes while maintaining or increasing quality is crucial to the success of the program. Information Assurance is a program facet that must be improved to help achieve the DoD training transformation vision and goals of speed, agility and security. Information Assurance testing is a necessary activity that must be successfully accomplished and documented before systems and networks can be accredited for operations. The Combat Air Force (CAF) Distributed Mission Operations (DMO) Network Cross Domain Solution (DCDS) test engineers employed automated test tools in conjunction with a structured test methodology to reduce the Information Assurance testing effort. Streamlining the Information Assurance testing process with automation allowed the program to increase test coverage and build confidence in the system's security posture while reducing the time to execute tests, analyze results, and generate test reports.

This paper describes the automated test tools and related processes that were successfully implemented to support Information Assurance testing of the DCDS. The paper provides objectives to consider during the selection and implementation of automated test tools and addresses process enhancements implemented to establish and sustain a high level of security assurance required in the warfighter's integrated live, virtual, and constructive training environments.

OVERVIEW

The CAF DMO DCDS Team is responsible for conducting Information Assurance testing to support Certification and Accreditation of the DCDS using the Joint Army, Air Force, Navy (JAFAN) 6/3 Protection Level 3 requirements [1]. The criticality of the DCDS in protecting data warrants a high degree of security testing assurance. The DCDS is deployed to multiple sites around the world. Each deployment of the DCDS requires the DCDS Team to conduct separate formal Information Assurance tests in support of receiving an approval to operate the system at a particular site.

Early testing to support each DCDS deployment was conducted using 65 manual test cases consisting of a voluminous set of tedious, time consuming test steps. The test engineers executed the manual test cases in approximately 16 hours during formal testing. The first DCDS test was accomplished in 2007 and since then, improvements have been made as the testing process evolved. The latest and most dramatic improvements involved implementing automated test tools. The DCDS Team enhanced the Information Assurance test process using industry-proven techniques and leveraged automated tools to streamline the testing effort. The test suite currently consists of 30 manual test cases and 144 automated test cases. The test engineers now execute the full complement of tests in approximately 8 hours.

AUTOMATED TESTING OBJECTIVES

The DCDS Team established five key objectives as a basis for defining and implementing an effective automated testing approach (see Table 1).

Table 1. Test Automation Key Objectives

1. Leverage existing industry-accepted security testing tools
2. Allow for customized automated test cases to be developed and executed
3. Streamline the security test execution process
4. Generate detailed and understandable test artifacts
5. Maintain requirements traceability within testing artifacts

The major consideration that applies to each key objective listed above is cost. Investing in test automation can be an expensive undertaking that runs a high risk of completely eliminating any overall cost savings and test assurance gains established as program goals. Numerous inexpensive or free automated

security test tools exist. However, they may not sufficiently support all test automation key objectives in Table 1. Conversely, there are automated test tools that implement a rich feature set, but they tend to be very expensive. Initial investment costs to consider include the tool's purchase price as well as expenses related to: (1) additional hardware and software needed for the tool to function; (2) vendor technical training; (3) reduced initial productivity resulting from "the learning curve"; and (4) vendor technical support. Recurring costs include renewing the annual license and the time and effort needed to develop, tune, and finalize automated tests and periodically update (or fix) automated tests to align with evolving system baselines. The decision to implement any automated test techniques should undergo an in-depth, return-on-investment analysis before committing to such an under taking. Return-on-investment should be assessed based on direct and indirect benefits across people, processes, and technology drivers [2].

A certain level of automation can be productively implemented on most programs to benefit the Information Assurance testing process. Figure 1 depicts the test components that comprise the full DCDS security test suite. The test suite used in conjunction with a test process adapted for test automation achieved a 50% reduction in the time to formally execute DCDS Information Assurance tests and increased assurances that the system meets established security requirements by conducting more comprehensive and exhaustive tests.

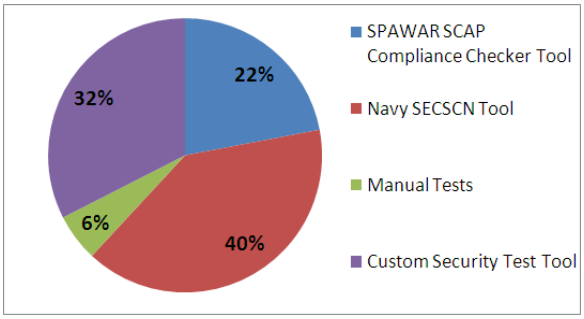

Figure 1. DCDS Test Suite Composition

The following paragraphs provide details explaining how each of the test automation key objectives were considered and used by the DCDS Team to implement an automated test process.

Objective #1 - Leverage Existing Test Tools

Automated security tools are invaluable for Information Assurance testing efforts since they can perform a bulk of the verification tests that must be accomplished to support system Certification and Accreditation. Selection of security test tools should be primarily based on the tool's compatibility with the system architecture, security requirements verification coverage, and the program's cost and schedule budgetary constraints.

Coordination with the security stakeholders is also important to ensure the selected toolset aligns with the program's security testing objectives and effectively supports the overall Certification and Accreditation effort. It is counterproductive to select a test tool that either underachieves (e.g., falls well short of meeting the established security test objectives) or overachieves the program's objectives (e.g., includes expensive unnecessary features or primarily conducts security verification checks beyond what the Designated Accrediting Authority representatives require for Certification and Accreditation). A multitude of commercial, shareware, and DoD security testing tools exist ranging from those that scan for common security vulnerabilities to those that analyze systems for compliance with specific organizational security policies, such as the DoD Information Systems Agency (DISA) Gold Disk [3].

Recently, the National Institute of Standards and Technology (NIST) established a standard protocol to help the Government overcome the challenges of validating compliance with applicable technical security requirements. NIST is a federal technology agency that works with industry to develop and apply technology, measurements, and standards. NIST Special Publication (SP) 800-117 addresses the Security Content Automation Protocol (SCAP™) which is a multi-purpose protocol used to perform automated configuration, vulnerability, and patch checking, technical control compliance activities, and security measurement [4]. SCAP-enabled tools are available that can be used to perform automated security configuration verification checks, and many have been validated by NIST (visit the NIST SCAP Validated Products website [5] for the current list). The configuration checks performed by SCAP-enabled tools are expressed as SCAP content, which are machine-readable eXtensible Markup Language (XML) policy documents. One example of a NIST-validated SCAP-enabled tool is the Center for Internet Security (CIS) Configuration Assessment Tool (CAT) which is available exclusively to CIS Security Benchmark members [6]. The CIS-CAT has the ability to perform configuration checks against a large set of CIS security benchmarks existing as SCAP content developed and maintained by CIS.

DISA and NIST are conveying and publishing the testable portions of security configuration guides as SCAP content. DISA provides global NetCentric solutions for warfighter support. Publically available SCAP content is currently only limited to Microsoft Windows® operating systems and Microsoft Internet Explorer® web browsers [7]. MITRE maintains a large, open source collection of vulnerability SCAP content in a publicly accessible repository (visit the MITRE OVAL repository web site [8] for details). The vulnerability content is ingestible by SCAP-enabled tools to allow automated checking for known security vulnerabilities associated with common applications and products. This provides significant value to the DoD by providing a capability for organizations to automatically identify vulnerabilities frequently exploited to harm networked systems.

Unfortunately, there are recognized cases where SCAP content does not work correctly with SCAP-enabled tools. This problem demonstrates the current lack of maturity of SCAP and the SCAP-enabled tools [7]. However, it should not preclude a programs' motivation to consider the use of SCAP-enabled tools for conducting automated Information Assurance testing since a large percentage of tests will work correctly and the few (if any) erroneous tests that might exist can be disabled or fixed.

The DCDS Team chose to implement the Navy security scanner tool called SECSCN to test the Linux servers because it generates test artifacts based on Director Intelligence Community Directive 6/3 [9], which contains security requirements identical to JAFAN 6/3 with a few minor exceptions. The DCDS Team also implemented the Space and Naval Warfare (SPAWAR) SCAP Compliance Checker tool which is free for use by any Federal Government employee or contractor. The DCDS Team obtained the DISA SCAP Automated Benchmark for Windows [10] as input to the SCAP Compliance Checker to automate testing of the Microsoft Windows systems. Using the DISA SCAP Automated Benchmark precluded the need to perform the time-consuming manual steps normally required as part of executing the DISA Gold Disk. The DCDS Team augmented the DCDS security test suite with a custom developed tool that performed tests not covered by SECSCN and the SCAP Compliance Checker. The custom security tool also allowed DCDS-specific automated tests to be developed and executed. Additional details regarding the custom tool are addressed in Objective #2 – Create Customized Automated Security Checks.

For systems that employ a large amount of security-relevant GUI-based functionality, it may be necessary to use GUI-based automated test tools. If the program decides to implement a GUI-based test tool for functional testing, it would be wise to leverage the tool to support Information Assurance testing when feasible. The DCDS does not employ a significant set of GUI-based security functionality. Therefore, the DCDS Team determined that implementing an automated GUI tool would be an unjustified program investment since the automated test cases implemented by the automated toolset encompass over 94% of the security test cases.

Programs should be prepared to handle common pitfalls associated with security test tools. It is not uncommon for test tools to report false-positive findings (i.e., discrepancies associated with security configuration settings which are not relevant to how the system is designed for operations). The DCDS Team experienced SCAP related issues that required mitigation. For example, a few configuration checks specified within DISA SCAP content resulted in unacceptable execution times (e.g., an individual test case took over eight hours to complete). Also, there were instances where checks specified in Mitre's open-source vulnerability SCAP content resulted in false-positive findings caused by incorrect logic performed by the test tool. The DCDS Team handled these problems as follows: (1) developed rationale explaining why false-positive findings are not applicable or are not considered security risks so test results can be defended during the Certification and Accreditation process; (2) changed the SCAP content to disable faulty checks; (3) updated the SCAP content to resolve discrepancies; and (4) developed custom automated test cases to replace faulty checks (see Objective #2 – Create Customized Automated Security Checks).

Objective #2 – Create Customized Automated Security Checks

The goal of this objective is to minimize the amount of onerous, time-consuming manual test cases needed to accomplish Information Assurance testing by leveraging customized automated tests. Many security test tools perform generic verification checks designed primarily for security-enabled products deployed in commonly used environments. While most of these generic checks are useful for verifying compliance with applicable security requirements, the checks typically cover a subset of the overall set of verification checks that are necessary for successful Certification and Accreditation. Some tools include steps that instruct the user to perform a multitude of manual checks. Also, many test tools, such as SECSCN, do not provide a capability to effectively create custom test cases to

verify system-specific configurations and functionality. Therefore, a key aspect of achieving this goal is to select and implement an automated test tool that provides the ability to create and modify custom, system-specific test cases.

Test tools are emerging that offer robust, innovative features to allow custom security verification checks to be built and executed more quickly, easily, and cheaper than repetitively having to execute manual steps. An example is SCAP-enabled test tools. SCAP content ingested and used by test tools can be tailored to align with unique aspects of the system, and system-specific security configuration checks can be incorporated into SCAP content as additional configuration checks performed by the tool. It is very important to closely track and manage all modifications made to third party SCAP content. At a minimum, the following should be performed by the program's technical staff when changes are made to third party SCAP content: (1) identify the specific changes and coordinate the changes with the security stakeholders to receive concurrence; (2) control and manage the changes through the program's Configuration Management process; and (3) implement a mechanism to merge the changes into newly acquired SCAP content released by third parties to ensure the changes persist within new versions.

The test tool developed by the DCDS Team implements a unique grammar for constructing human-readable, custom automated tests for the network devices and Linux platforms within the DCDS, including the security-enabled applications installed on the Linux platforms. The philosophy behind the tool is to create a simple, human readable test plan (including test case procedures or test steps) that feed directly into the test tool. This allowed the security stakeholders the ability to fully understand exactly how requirements were interpreted and verified during Information Assurance testing.

Objective #3 – Streamline Test Execution

Many system deployment timelines are established within an aggressive schedule. Therefore, each hour that can be saved conducting formal testing is an hour that can be productively used on other activities. This objective involves ensuring test execution occurs in a smooth, logical, and efficient manner. The ultimate goal for the DCDS Team was to reduce the schedule footprint needed for formal security configuration testing, which includes executing the tests, gathering test results data, and analyzing test results to identify potential security deficiencies. This objective was achieved using the DCDS automated security tool suite. The automated test tools coupled with enhanced test processes allowed a six-fold increase in test cases while reducing the overall time by 50% for conducting the tests and analyzing the results.

Depending on the size of the system infrastructure, executing automated tests (to include, as necessary, uploading the tests, manually initiating automated test execution, collecting test results, analyzing test results, and removing the test ruminants) can be a daunting effort, but clearly not as much if the tests were manual. Some system infrastructures containing a large number of components may warrant the need to employ enterprise management tools such as BMC BladeLogic Automation Suite (a SCAP-enabled toolset) [11] or Microsoft® System Center Configuration Manager (with the SCAP Extensions Module) [12]. These types of tools can be used to automatically execute security configuration verification checks from a central location, and collect and consolidate the results at the enterprise level. Enterprise management tools can be leveraged to conduct security testing across the entire infrastructure, thereby reducing the overall effort needed to individually execute security tests on each component of the system's infrastructure. Additionally, the tools can be configured to periodically validate that the infrastructure continues to comply with established security configuration standards during operation.

For programs with mid-sized to large system infrastructures that are not interested in deploying a commercial enterprise management solution, an option to consider is to implement Capistrano [13] to execute automated security tests across components of the infrastructure. Capistrano is an open source Unix-based utility for executing commands in parallel on multiple systems, primarily for deploying applications. Capistrano can be used to deploy automated test utilities or scripts onto multiple servers, initiate the execution of the automated tests, collect the test results into a single centralized location, and remove the test-related programs and results files from each system, as necessary.

Objective #4 - Generate Detailed and Understandable Security Test Artifacts

An important aspect of this objective is for the automated test tools to: (1) clearly articulate, in an understandable manner, the security deficiencies identified by the tools; (2) identify requirements affected by security deficiencies; and (3) produce deliverable-level test reports with minimal human intervention. Achieving this objective fosters an ability to generate security test reports that contain relevant

evidentiary information needed for system Certification and Accreditation. The automated test tools used by the DCDS Team fully satisfy this objective.

The SECSN tool automatically generates a detailed test report in Hyper Text Markup Language (HTML) format that explains identified security deficiencies in an understandable manner along with the affected requirements, and provides specific supporting details such as the command used to assess the system configuration and rationale explaining why the configuration is important. Another valuable test artifact generated by SECSCN is a Security Requirements Traceability Matrix which maps each test case to the applicable source requirement.

The custom tool developed by the DCDS Team provides a comprehensive set of test artifacts used to support Certification and Accreditation. Figure 2 depicts an example of a failed check within the test report. The test report includes the source requirement, component allocations, design criteria, test case description, test step identifier, automated test steps (in human understandable form) and a pass/fail designation. Since the custom tool does not have a pedigree comparable to the SECSCN tool, the report includes specific details explaining why each verification check either passed or failed. These details include a pointer to the specific line within the configuration file that caused the check to either pass or fail. If required, confidence can be gained that the tool is reporting the correct results by following the pointers contained in the test report and manually verifying the results support compliance (or failure) with the requirement.

Requirement	4.B.3.a(9)(e) - The following must be specified: Aging of static authenticators.
Server	Server Name
Component	Operating System Name and Version
Design Criteria	The system must force user passwords to be changed every 180 days
Automated Test	(1)NE-0001 (1) /etc/login.defs (2)PASS_MAX_DAYS = 180 (3)(END)end_listing
P/F	Fail
Failure Reason	Expected: PASS_MAX_DAYS = 180 Actual: PASS_MAX_DAYS = 365 --> line 17
Passed Checks	Expected: NE-0001 (1) /etc/login.defs Actual: Block <start> --> line 1; <end> --> line 24 Actual: NE-0001 (1) /etc/login.defs --> line 1

Figure 2. Example Failed Check Test Report

The SCAP Compliance Checker tool generates reports in HTML or XML format. Figure 3 depicts a snapshot of HTML-formatted test results information generated

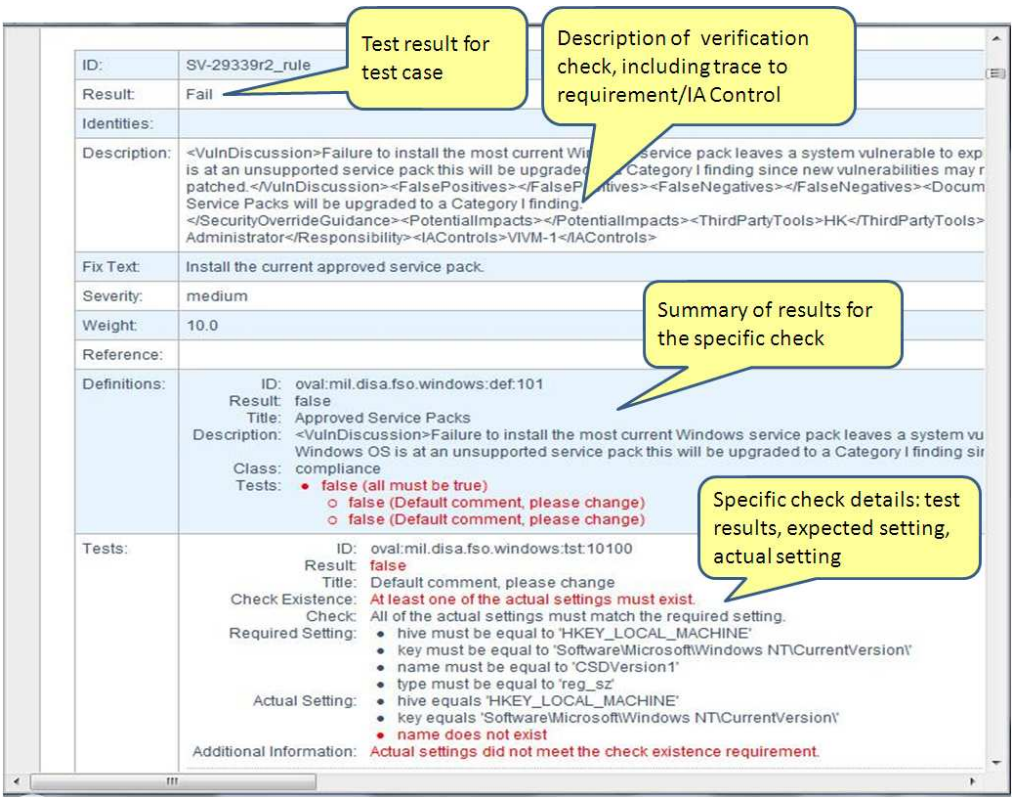

Figure 3. Example SCAP Compliance Checker Report

for a single verification check. The tool generates reports in sufficient detail to understand the specific verification check and the associated test results. However, there were some ambiguous test results that required supplemental data contained in the applicable DISA Security Technical Implementation Guide [14] in order to fully understand how to resolve the finding. The SCAP Compliance Checker tool was able to ingest the DISA SCAP Automated Benchmark for Windows and execute the set of specified automated tests, but it was not fully compatible with the DISA SCAP content. For example, the "Description" part of the report includes XML tags that were not correctly formatted within the report. This incompatibility only presented a minor challenge in reviewing the verification check description information and did not affect execution of the automated tests.

For tools that generate test reports in HTML or XML format, programs can leverage the report formats to perform custom, post test execution processing to generate higher fidelity test reports. For example, the test reports can be ingested into a database, and custom-developed utilities can process the data to perform the following: (1) generate trending and metrics reports; (2) filter false-positive findings from test reports; (3) map untraced test results to the affected security requirements; and (4) maintain and associate risk management data to each discrepancy (e.g., Plan of Action and Milestones information).

Objective #5 - Maintain Requirements Traceability Information

Requirements are the foundation for any system, forming the basis of the system's design and implementation. The most important aspect of Information Assurance testing is ensuring the system complies with all applicable security requirements and the resultant test evidence is sufficient to support system Certification and Accreditation. Without the aid of an automated tool, the effort to maintain requirements traceability throughout the system lifecycle is a daunting task. The ability to trace requirements and ensure requirements coverage is an important feature of the automated security testing toolset used by the DCDS Team. Equally important is the ability to capture meaningful engineering data such as design criteria (often referred to as detailed or derived requirements) and descriptions of the system security architecture implemented to comply with requirements. Integration of these types of engineering artifacts within the toolset is paramount for providing security stakeholders the ability to better understand the "big picture" with regards to the overall security design and how the system was tested to verify compliance with security requirements. Security stakeholders include developers, integrators, test engineers, and the government Designated Accrediting Authority representatives. Figure 4 depicts an example illustrating the value of maintaining traceability information from requirements through test artifacts.

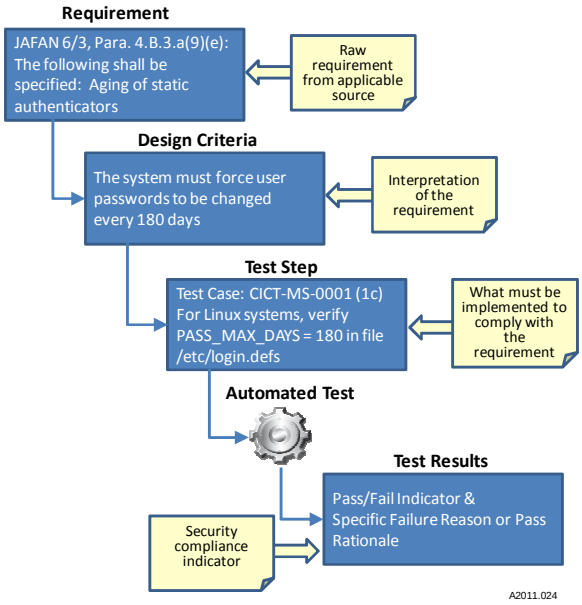

Figure 4. Security Engineering Traces

A variety of different Commercial-Off-The-Shelf products exist that can be implemented to maintain and manage traceability data such as IBM Rational DOORS® [15]. However, there are very few requirements management products that can be implemented out-of-the-box to directly link traceability data to automated test artifacts. For products that provide end-to-end traceability such as Hewlett Packard Quality Center [16], the testing features offered by the tool (or the integrated add-on products) may not meet the program's automated testing objectives. Development of custom middleware to incorporate data maintained by traceability tools with automated test artifacts is a technically feasible alternative. Any product that stores information within a non-proprietary database or maintains (or generates) data in common formats such as XML can be integrated into a quasi-seamless toolset. Programs have successfully implemented custom middleware utilities to bind disparate tools together into a fully integrated end-to-end toolset, although this effort could add considerable cost to the program [17].

SCAP-enabled tools can be used to manage and maintain requirements mappings to specific verification checks performed by the tool. SCAP

content allows requirements mappings to be characterized such that each security configuration check is traced to the applicable source requirements. However, SCAP content may not pre-exist to specifically cover the system's source security requirements. Since SCAP content is in XML format, it is possible to enhance the SCAP content to incorporate the applicable requirements mappings that are needed to support the system's Certification and Accreditation effort. Another automation-related alternative is for the test engineers to ingest test reports into a database that appropriately maps the test results to the system's source requirements.

The SECSCN tool and the custom security test tool allow the traceability information in Figure 4 to be effectively maintained. For the requirements traces associated with the SCAP Compliance Checker tool, the DCDS Team maintained a hardcopy version of mappings from the test ID contained within the DISA Windows SCAP content (used as input for the tool) and the DCDS source requirements from JAFAN 6/3.

PROCESS IMPROVEMENTS

Implementation of automated test tools was an important factor in improving the quality of the security testing process since it allowed numerous, comprehensive tests to be accomplished quicker than a human can perform the tests while reducing the risk of human error. Equally important was the need to adapt the existing security test process to accommodate the use of automated tools and then actively strive to implement regular enhancements to further improve the process. The following paragraphs describe process improvements implemented by the DCDS Team while incorporating the use of automated test tools within the overall security testing methodology.

Automated Tests vs. Manual Tests

In an ideal scenario, all security tests required for Certification and Accreditation would be conducted using automated tests. Unfortunately, it is impractical to expect the full set of security tests to be performed solely using automation, no matter how robust the suite of testing tools [18]. Some test cases can be cost prohibitive to fully automate. For example, system-level or end-to-end security testing typically involves a structured sequence of test steps to verify functionality implemented by multiple system components operating in harmony. It is difficult to effectively automate these types of tests due to technical constraints of being able to align the sequence of test steps across multiple system components or because the time to implement is considered exorbitant. The DCDS Team established guidelines to assist in deciding which DCDS security tests to automate (see Table 2). The criteria within Table 2 helped focus the DCDS Team on automating test cases based on cost and value and avoid automating tests with minimal return on investment.

Table 2. Manual vs. Automate Criteria

Criteria	Automate	Manual
Involves conducting a test step that can be accomplished using the operating system's command line	✓	
Involves a tedious verification check potentially prone to human error/oversight	✓	
Candidate for recurring test (regression test, supports "test early & often" principle)	✓	
Ratio of time to manually execute over time to automate is <= 4%	✓	
Good "checks-and-balances" test (e.g., verify CM is functioning as expected, verify the installation process worked correctly)	✓	
Ratio of time to manually execute over time to automate is > 4%		✓
Involves viewing or executing functions contained within User Interface screens		✓
Automating raises an initial concern of technical feasibility		✓
Involves a structured sequence of test steps to verify functionality implemented by multiple system components operating in harmony		✓
Automated test will be difficult to maintain/update for subsequent baselines		✓
Test involves some level of human intervention (human judgment required)		✓

Test Early and Often

Aggressive schedules, limited resources, and the challenges of developing complex systems sometimes drive programs to implement test methodologies that entail generating test procedures in parallel with the system development and integration effort and then testing the system for the first time towards the latter part of the schedule. The system is frequently baselined just-in-time to execute the full gamut of test procedures during a planned dry-run event. Dry-run testing is typically used to ensure the system is stable and to rehearse for the formal test event. It is not unusual for dry-run testing to immediately precede the formal test event with very little slack time between the two activities. This approach adds risk to the schedule since it leaves minimal time to correct discrepancies identified during dry-run testing.

The DCDS Team instituted a "test early and often" approach as depicted in Figure 5 to reduce the risk of significant system vulnerabilities existing prior to formal security testing [19]. The philosophy behind "test early and often" is to construct individual

automated security tests as early as possible during the development and integration phase. Each completed automated test is added to a growing collection of automated test cases. The methodology involves continuously building the collection of automated test cases, executing the tests regularly during development and integration, tuning the test cases as necessary, and resolving any unexpected discrepancies as early as possible.

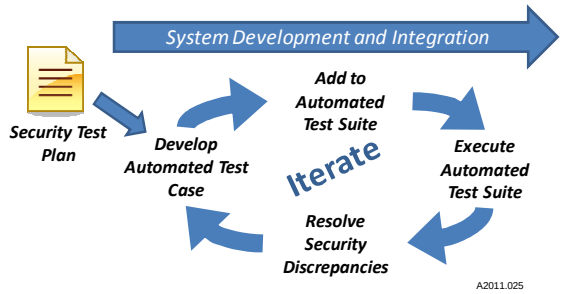

Figure 5. Test Early and Often Methodology

The Navy SECSCN tool and the SPAWAR SCAP Compliance Checker tool were used to support the "test early and often" process. Also, the custom test tool implemented by the DCDS Team facilitated the "test early and often" approach since the development of automated test cases using the tool followed a similar process typically used to construct manual test procedures. More specifically, the tool accepted input in the form of a logically constructed test procedure that was human readable and understandable, similar to the manner in which manual test procedures are developed. Once the test procedure is completed using the tool, it can be immediately executed to support the "test early and often" methodology.

The "test early and often" methodology cannot be effectively accomplished using a large set of manual tests that are tedious, time consuming, and prone to human error. It is not reasonable to expect manual testing to be accomplished at the frequency required to effectively implement a "test early and often" methodology. However, periodically executing the manual security tests at strategic times during the development and integration phase must be accomplished to realize the full benefits of this methodology.

The "test early and often" methodology used by the DCDS Team insured compliance with all applicable security requirements prior to the system baseline milestone. The process identified incorrectly configured security settings during development and integration, and recognized instances where system evolution caused regressions within the system's security posture. Since these problems were identified early, it was possible to fix the problems well in advance of the system baseline milestone.

Simplify Manual Security Testing

The robustness of the automated test suite allowed the DCDS Team to establish a streamlined process to reduce the manual testing effort on systems tested after the first deployment. This test process involves executing a subset of the overall test cases to verify the system continues to function as expected while maintaining a high degree of testing assurance. Although the streamlined test approach is largely feasible because the same baseline is deployed to multiple sites, it is possible to employ a similar process for verifying system changes made as part of a structured patch management process involving minor to moderate system changes.

The DCDS Team used the following three test categories to support the streamlined test process:

- Demonstration Tests – These manual tests demonstrate certain DCDS functionality (e.g., verify that when a user executes a specific function, the application generates the expected output). This category of test is normally executed only once to support formal Certification and Accreditation, but must also be performed when a system change warrants specific tests to be performed on subsequent baselines. The DCDS Team established Re-Execution Criteria defining specific conditions warranting performance of particular Demonstration tests during security testing of subsequent baselines.
- Configuration Tests – These manual and automated tests involve verification of system security configuration settings. This category of test is performed as part of each formal security test activity to ensure all security-relevant configuration settings are correct. This test is especially important for verifying that the manual installation steps, which are prone to human error, were properly conducted
- Check-Out Tests – These manual tests are a subset of the Demonstration Tests established to verify that the security-critical interfaces and system process dependencies are properly enabled and implemented after installation (e.g., verify that data generated and transmitted from one system is properly received and processed by another system). This category of test is performed as part of each formal security test

activity to ensure the system is functioning correctly end-to-end.

Security testing of the first deployment of a DCDS baseline involves conducting all the automated and manual tests to support the Certification and Accreditation activities needed for an Approval to Operate. For subsequent DCDS deployments to different sites, the DCDS Team executes only the Configuration and Check-Out tests. This reduces the manual test cases that need to be executed by approximately 50%.

The following aspects greatly contribute to the program's ability to implement a streamlined security test approach similar to the approach used by the DCDS Team:

- The program implements a structured, sound Configuration Management program to control, track, and manage system changes;
- The effect each system change has on the security posture of the system can always be positively determined (and defended);
- The system installation process is highly structured, stable, and repeatable (e.g., there is a high degree of assurance that the installation process will not introduce new discrepancies into the deployed system); and
- The program worked directly with the program's Designated Accrediting Authority to ensure system Certification and Accreditation can be maintained while employing a streamlined security test approach.

Perform Checks-and-Balances

Automated security test tools typically verify the system's operational-level technical mechanisms and are normally not used to verify development processes that are essential in preserving the system's security posture during development and integration. Configuration Management problems can negatively impact system quality, delay deployments, and increase system development and lifecycle costs [20]. The DCDS requires a high degree of security assurances which could be tarnished if problems with Configuration Management are ever encountered.

The DCDS Team realized the importance of maintaining the proper level of security assurance throughout the entire DCDS life cycle and decided to leverage the rich feature set implemented by the automated tool suite to perform checks-and-balances on programmatic aspects, such as Configuration Management. The DCDS Team built detailed test cases designed to verify the integrity of security-relevant configuration items such as the installed Red Hat Enterprise Linux packages and versions and application configuration files. For example, instead of inspecting individual parts of a configuration file, the DCDS Team constructed tests to inspect each-and-every line of the file to ensure unexpected changes were not made to the baseline. Augmenting the automated tests to perform a feasible level of checks-and-balances was a relatively simple and inexpensive effort that greatly increased the overall confidence gained with the stakeholders that the DCDS is being properly controlled during development and correctly installed during deployment.

CONCLUSION

A one-size-fits-all solution for implementing an automated test capability is not feasible; the solution must be customized to help achieve the program's unique goals. Investing in test automation must be carefully considered based on a return-on-investment analysis tailored specifically for the program. The DCDS Team implemented two different DoD security test tools augmented with an innovative custom test tool to successfully conduct Information Assurance testing of the DCDS. The test cases increased six-fold and the overall test schedule was trimmed in half when compared to the previous manual-only testing methodology. Employing automated test tools allowed the engineers to institute a "test early and often" approach which was invaluable to the program for identifying and correcting security deficiencies well in advance of the system baseline milestone.

In summary, employing automated test tools in conjunction with a process that aligns with the use of automated tools is a practical approach for streamlining the security testing process. The potential benefits include increased test assurances, reduced formal testing timelines, and increased confidence the system will successfully achieve Certification and Accreditation.

REFERENCES

[1] Department of Defense (2004), Joint Air Force, Army, Navy (JAFAN) 6/3 Manual (FOUO)

[2] Keene an NTT DATA Company (2006), Whitepaper: ROI on Test Automation, [http://www.keane.com/resources/pdf/WhitePapers/WP_ROIforTestAutomation.pdf]

[3] Defense Information Systems Agency (2011), Gold Disk Web Page, [http://iase.disa.mil/stigs/gold_disk/index.html], and DoD General Purpose STIG Checklist and Tool Compilation CD Web Page, [http://iase.disa.mil/stigs/dod_purpose-tool/index.html]

[4] National Institute of Standards and Technology (2010), NIST Special Publication 800-117 (2010), Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0, [http://csrc.nist.gov/publications/nistpubs/800-117/sp800-117.pdf]

[5] National Institute of Standards and Technology (2011), Security Content Automation Protocol Validated Products Website [http://nvd.nist.gov/scapproducts.cfm]

[6] The Center for Internet Security (CIS), CIS Configuration Assessment Tool (CIS-CAT) Datasheet, [http://benchmarks.cisecurity.org/en-us/docs/collateral/cis-cat_datasheet.pdf]

[7] Verder Pol, J (2010), SCAP Compliance Checker: Developing a Government-Funded SCAP-Validated Application, [http://scap.nist.gov/events/2010/itsac/presentations/day2/Innovative_Uses_of_SCAP-Developing_a_Government-Funded_SCAP-Validated_Application.pdf]

[8] MITRE (2011), OVAL Repository Website [http://oval.mitre.org/repository]

[9] Director of Central Intelligence (2003), Director of Central Intelligence Directive (DCID) 6/3 Manual, Protecting Sensitive Compartmented Information Within Information Systems

[10] Defense Information Systems Agency (2011), Security Technical Implementation Guide (STIG) Operating Systems Web Page, [http://iase.disa.mil/stigs/os/index.html#]

[11] BMC Software (2011), BMC BladeLogic Automation Suite Datasheet, [http://documents.bmc.com/products/documents/32/50/203250/203250.pdf]

[12] Microsoft® (2006), System Center Configuration Manager 2007 Desired Configuration Management Web Page, [http://www.microsoft.com/systemcenter/en/us/configuration-manager/cm-desired-configuration-management.aspx]

[13] GitHub, Inc © (2011), [https://github.com/capistrano/capistrano/wiki]

[14] Defense Information Systems Agency (DISA) Security Technical Implementation Guides (STIG) Web Page (2011), [http://iase.disa.mil/stigs/index.html]

[15] IBM® Corporation, IBM Rational® DOORS® (2010), [http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=PM&subtype=SP&appname=SWGE_RA_RA_USEN&htmlfid=RAD14037USEN&attachment=RAD14037USEN.PDF]

[16] Hewlett-Packard Development Company (2011), HP Quality Center Software Web Page, [http://www8.hp.com/us/en/software/software-product.html?compURI=tcm:245-937045&pageTitle=quality-center]

[17] Dustin, E (2001), Lessons in Test Automation: A Manager's Guide to Avoiding Pitfalls When Automating Testing, [http://www.informit.com/articles/article.aspx?p=21467]

[18] Hoffman, D (1999), Cost Benefits Analysis of Test Automation, [http://www.softwarequalitymethods.com/Papers/Star99 model Paper.pdf]

[19] McGregor, J (2007), Test Early Test Often, Journal of Object Technology, Vol. 6, No. 4, [http://www.jot.fm/issues/issue_2007_05/column1/]

[20] Aberdeen Group (2007), The Configuration Management Benchmark Report, Formalizing and Extending CM to Drive Quality, [http://www.isscorp.com/Configuration_Management_Final.pdf]