Software Engineering Institute

Source Code Analysis Laboratory (SCALe) for Energy Delivery Systems

Robert C. Seacord
William Dormann
James McCurley
Philip Miller
Robert Stoddard
David Svoboda
Jefferson Welch

December 2010

TECHNICAL REPORT
CMU/SEI-2010-TR-021
ESC-TR-2010-021

CERT® Program
Unlimited distribution subject to the copyright.

http://www.cert.org

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2010 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use. This document may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Figures

List of Tables

Acknowledgments

We would like to acknowledge the contributions of the following individuals to the research presented in this paper: Doug Gwyn, Paul Anderson, William Scherlis, Jonathan Aldrich, Yekaterina Tsipenyuk O'Neil, Roger Scott, David Keaton, Thomas Plum, Dean Sutherland, Timothy Morrow, Austin Montgomery, and Martin Sebor. We would also like to acknowledge the contribution of the SEI support staff and management, including Paul Ruggiero, Shelia Rosenthal, Terry Ireland, Sandra Brown, Michael Wright, Archie Andrews, Bill Wilson, Rich Pethia, and Clyde Chittister. This research was supported by the U.S. Department of Energy (DOE) and the U.S. Department of Defense (DoD).

Abstract

The Source Code Analysis Laboratory (SCALe) is an operational capability that tests software applications for conformance to one of the CERT® secure coding standards. CERT secure coding standards provide a detailed enumeration of coding errors that have resulted in vulnerabilities for commonly used software development languages. The SCALe team at CERT, a program of Carnegie Mellon University's Software Engineering Institute, analyzes a developer's source code and provides a detailed report of findings to guide the code's repair. After the developer has addressed these findings and the SCALe team determines that the product version conforms to the standard, CERT issues the developer a certificate and lists the system in a registry of conforming systems. This report details the SCALe process and provides an analysis of energy delivery systems. Though SCALe can be used in various capacities, it is particularly significant for conformance testing of energy delivery systems because of their critical importance.

1 Introduction

A key mission of the U.S. Department of Energy's (DOE) Office of Electricity Delivery and Energy Reliability (OE), and specifically its Cybersecurity for Energy Delivery Systems (CEDS) program, is to enhance the security and reliability of the nation's energy infrastructure. Improving the security of control systems that enable the automated control of our energy production and distribution is critical for protecting the energy infrastructure and the integral function that it serves in our lives.

The Source Code Analysis Laboratory (SCALe) provides a consistent measure that can be used by grid asset owners and operators and other industry stakeholders to assess the security of deployed software systems, specifically by determining if they are free of coding errors that lead to known vulnerabilities. This in turn reduces the risk to these systems from increasingly sophisticated hacker tools.

1.1 Software Security

Software vulnerability reports and reports of software exploitations continue to grow at an alarming rate, and a significant number of these reports result in technical security alerts. To address this growing threat to the government, corporations, educational institutions, and individuals, systems must be developed that are free of software vulnerabilities.

Coding errors cause the majority of software vulnerabilities. For example, 64 percent of the nearly 2,500 vulnerabilities in the National Vulnerability Database in 2004 were caused by programming errors [Heffley 2004].

An interesting and recent example is Stuxnet, the first publicly known worm to target industrial control systems and take control of physical systems. Stuxnet included malicious STL (Statement List) code, an assembly-like programming language that is used to control industrial control systems, as well as the first-ever PLC (programmable logic controller) rootkit[1] hiding the STL code. It also included zero-day vulnerabilities,[2] spread via USB drives, used a Windows rootkit to hide its Windows binary components, and signed its files with certificates stolen from unrelated third-party companies.

Stuxnet uses a total of five vulnerabilities: one previously patched (Microsoft Security Bulletin MS08-067) and four zero-days. The vulnerability reported by MS08-067 could allow remote code execution if an affected system received a specially crafted remote procedure call (RPC) request. The code in question is reasonably complex code to canonicalize path names, for example, to strip out ".." character sequences and such to arrive at the simplest possible directory name. The coding defect allows a stack-based buffer overflow from within a loop. The loop inside the function walks along an incoming string to determine if a character in the path might be a dot, dot-dot,

[1] A rootkit is software that enables continued privileged access to a computer while actively hiding its presence from administrators by subverting standard operating system functionality or other applications.

[2] A zero-day vulnerability is a previously unknown vulnerability that is revealed in an exploit.

slash, or backslash. If it is, then the loop applies canonicalization algorithms. The bug occurs while calling a bounded function call:

```
_tcscpy_s(previousLastSlash, pBufferEnd - previousLastSlash, ptr + 2);
```

This is a violation of the *CERT® C Secure Coding* rule "ARR30-C. Do not form or use pointers or array subscripts that are out of bounds" and can be detected by the static analysis tools and techniques being implemented by CERT, part of Carnegie Mellon University's Software Engineering Institute, and deployed in SCALe.

CERT takes a comprehensive approach to identifying and eliminating software vulnerabilities and other flaws. CERT produces books and courses that foster a security mindset in developers, and it develops secure coding standards and automated analysis tools to help them code securely. Secure coding standards provide a detailed enumeration of coding errors that have caused vulnerabilities, along with their mitigations for the most commonly used software development languages. CERT also works with vendors and researchers to develop analyzers that can detect violations of the secure coding standards.

Improving software security by implementing code that conforms to the CERT secure coding standards can be a significant investment for a software developer, particularly when refactoring or otherwise modernizing existing software systems [Seacord 2003]. However, a software developer does not always benefit from this investment because it is not easy to market code quality.

1.2 SCALe

To address these problems, CERT has created the Source Code Analysis Laboratory (SCALe), which offers conformance testing of software systems to CERT secure coding standards.

SCALe evaluates client source code using multiple analyzers, including static analysis tools, dynamic analysis tools, and fuzz testing. CERT reports any deviations from secure coding standards to the client. The client may then repair and resubmit the software for reevaluation. Once the reevaluation process is completed, CERT provides the client a report detailing the software's conformance or nonconformance to each secure coding rule. The SCALe process consists of the sequence of steps shown in Figure 1.

® CERT is a registered mark owned by Carnegie Mellon University.

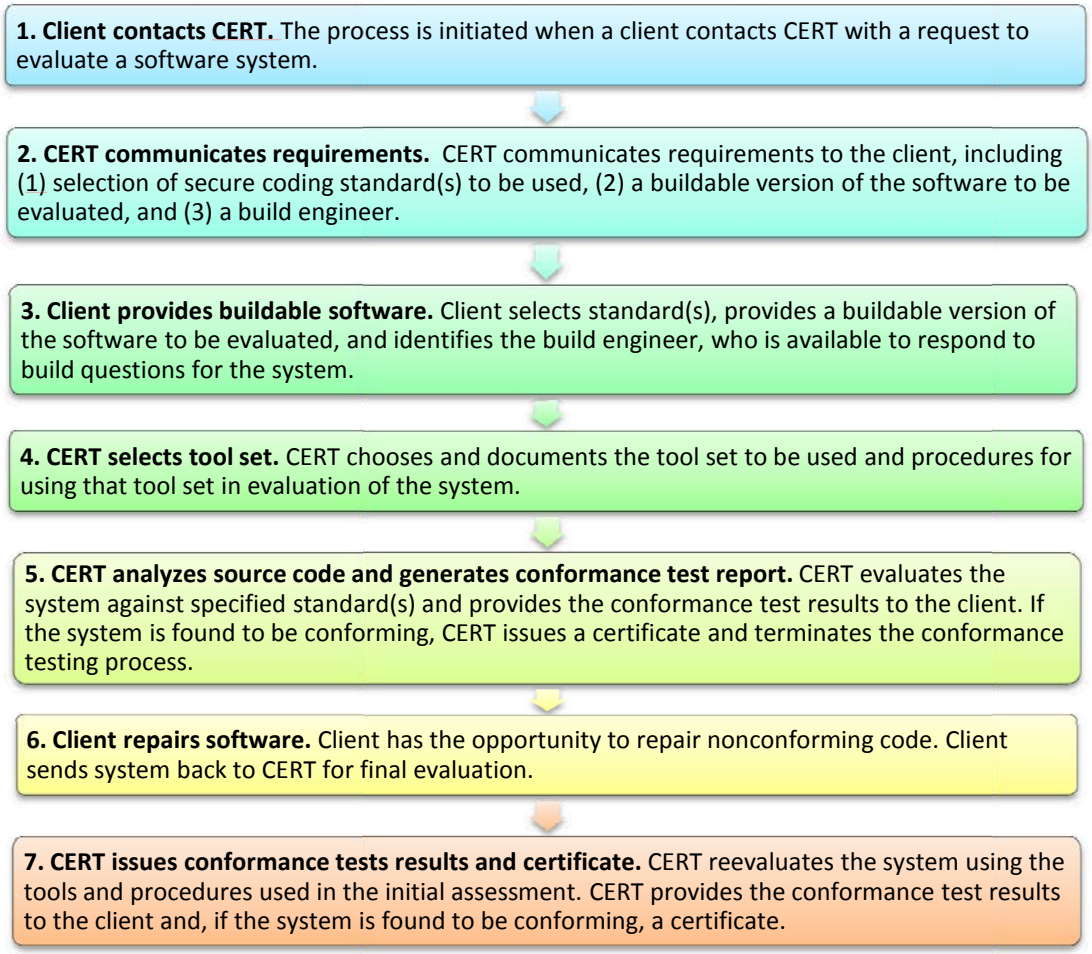

Figure 1: SCALe Process Overview

SCALe does not test for unknown code-related vulnerabilities, high-level design and architectural flaws, the code's operational environment, or the code's portability. Conformance testing is performed for a particular set of software, translated by a particular implementation, and executing in a particular execution environment [ISO/IEC 2005].

Successful conformance testing of a software system indicates that the SCALe analysis did not detect violations of rules defined by a CERT secure coding standard. Successful conformance testing does not provide any guarantees that these rules are not violated nor that the software is entirely and permanently secure. Conforming software systems can be insecure, for example, if they implement an insecure design or architecture.

Software that conforms to a secure coding standard is likely to be more secure than non-conforming or untested software systems. However, no study has yet been performed to prove or disprove this claim.

1.3 Conformance Assessment

SCALe applies *conformance assessment* in accordance with ISO/IEC 17000: "a demonstration that specified requirements relating to a product, process, system, person, or body are fulfilled"

[ISO/IEC 2004]. Conformance assessment generally includes activities such as testing, inspection, and certification. SCALe limits the assessments to software systems implemented in standard versions of the C, C++, and Java programming languages.

Conformance assessment activities are characterized by ISO/IEC 17000 [ISO/IEC 2004] as

- first party. The supplier organization itself carries out conformance assessment to a standard, specification, or regulation—in other words, a self-assessment—known as a supplier's declaration of conformance.
- second party. The customer of the organization (for example, a software consumer) performs the conformance assessment.
- third party. A body that is independent of the organization providing the product and that is not a user of the product performs the conformance assessment.

Which type of conformance assessment is appropriate depends on the level of risk associated with the product or service and the customer's requirements. SCALe is a third-party assessment performed by CERT or a CERT-accredited laboratory on behalf of the supplier or on behalf of the customer with supplier approval and involvement.

1.4 CERT Secure Coding Standards

SCALe assesses conformance of software systems to a CERT secure coding standard. As of year-end 2010, CERT has completed one secure coding standard and has three additional coding standards under development.

The CERT C Secure Coding Standard, Version 1.0, is the official version of the C language standards against which conformance testing is performed and is available as a book from Addison-Wesley [Seacord 2008]. It was developed specifically for versions of the C programming language defined by

- *ISO/IEC 9899:1999 Programming Languages — C*, Second Edition [ISO/IEC 2005]
- Technical Corrigenda TC1, TC2, and TC3
- *ISO/IEC TR 24731-1 Extensions to the C Library, Part I: Bounds-checking interfaces* [ISO/IEC 2007]
- *ISO/IEC TR 24731-2 Extensions to the C Library, Part II: Dynamic Allocation Functions* [ISO/IEC 2010a]

Most of the rules in *The CERT C Secure Coding Standard, Version 1.0*, can be applied to earlier versions of the C programming language and to C++ language programs. While programs written in these programming languages may conform to this standard, they may be deficient in other ways that are not evaluated by this conformance test.

It is also possible that maintenance releases of *The CERT C Secure Coding Standard* will address deficiencies in Version 1.0, and that software systems can be assessed against these releases of the standard.

There are also several CERT secure coding standards under development that are not yet available for conformance testing, including

- *The CERT C Secure Coding Standard, Version 2.0* [CERT 2010a]

- *The CERT C++ Secure Coding Standard* [CERT 2010b]
- *The CERT Oracle Secure Coding Standard for Java* [CERT 2010c]

1.5 Automated Analysis Tools

Secure coding standards alone are inadequate to ensure secure software development because they may not be consistently and correctly applied. Manual security code audits can be supplemented by automated analysis tools, including static analysis tools, dynamic analysis tools, tools within a compiler suite, and various testing techniques.

1.5.1 Static Analysis Tools

Static analysis tools operate on source code, producing diagnostic warnings of potential errors or unexpected run-time behavior. Static analysis is one function performed by a compiler. Compilers can frequently produce higher-fidelity diagnostics than analyzer tools, which can be used in multiple environments, because they have detailed knowledge of the target execution environment.

There are, however, many problems and limitations with source code analysis. Static analysis techniques, while effective, are prone to both false positives and false negatives. For example, a recent study found that more than 40 percent of the 210 test cases went undiagnosed by all five of the study's C and C++ source analysis tools, while only 7.2 percent of the test cases were successfully diagnosed by all five tools (see Figure 2) [Landwehr 2008]. The same study showed that 39.7 percent of 177 test cases went undiagnosed by all six of the study's Java code analysis tools and that 0 percent of the test cases were discovered by all six tools. Dynamic analysis tools, while producing lower rates of false positives, are prone to false negatives along untested code paths. The NIST Static Analysis Tool Exposition (SATE) also demonstrated that developing comprehensive analysis criteria for static analysis tools is problematic because there are many different perspectives on what constitutes a true or false positive [Okun 2009].

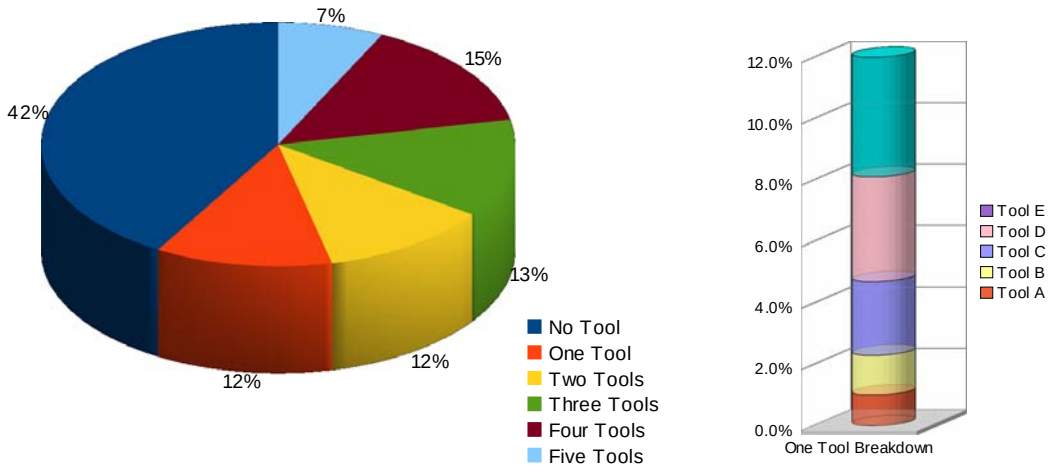

Figure 2: C and C++ "Breadth" Case Coverage [Landwehr 2008]

To address these problems, CERT is working with analyzer vendors and with the WG14 C Secure Coding Guidelines Rules Group (CSCR SG) to precisely define a set of analyzable secure coding

guidelines for the C99 version of the C Standard [ISO/IEC 2005], as well as for the emerging C1X major revision to the C standard [Jones 2010]. Having such a set of guidelines and standardizing them through the ISO/IEC process should eliminate many of the problems encountered at the NIST SATE and also increase the percentage of defects found by more than one tool. CERT is working on tools to support the set of analyzable secure coding guidelines. First, CERT is coordinating a test suite, under a Berkeley Software Distribution (BSD)-type license,[3] that will be freely available for any use. This test suite can be used to determine which tools are capable of enforcing which guidelines and to establish false positive and false negative rates. Second, CERT has extended the Compass/ROSE tool,[4] developed at Lawrence Livermore National Laboratory, to diagnose violations of the *CERT Secure Coding Standards* in C and C++.

1.5.2 Dynamic Analysis and Fuzz Testing

Dynamic program analysis analyzes computer software by executing that software on a real or virtual processor. For dynamic program analysis to be effective, the target program must be executed with test inputs sufficient to produce interesting behavior. Software testing techniques such as fuzz testing can stress test the code [Takanen 2008], and code coverage tools can determine how many program statements have been executed.

1.6 Portability and Security

Portability and security are separate, and sometimes conflicting, software qualities. Security can be considered a measure of *fitness for use* of a given software system in a particular operating environment, as noted in Section 1.2. Software can be secure for one implementation and insecure for another.[5]

Portability is a measure of the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE Std 610.12 1990]. Portability can conflict with security, for example, in the development of application programming interfaces (APIs) that provide an abstract layer over nonportable APIs while cloaking underlying security capabilities. Portability can become a security issue when developers create code based upon a set of assumptions for one implementation and port it, without adequate verification, to a second implementation where these assumptions are no longer valid. For example, the C language standard defines a *strictly conforming* program as one that uses only those features of the language and library specified in the standard [ISO/IEC 2005]. Strictly conforming programs are intended to be maximally portable among conforming implementations. Conforming programs may depend upon nonportable features of a conforming implementation.

Software developers frequently make assumptions about the range of target operating environments for the software being analyzed:

- The null pointer is bitwise zero. This assumption means that initializing memory with all-bits-zero (such as with `calloc`) initializes all pointers to the null pointer value.

---

[3] http://www.opensource.org/licenses/bsd-license.php

[4] http://www.rosecompiler.org/compass.pdf

[5] An implementation is “a particular set of software, running in a particular translation environment under particular control options, that performs translation of programs for, and supports execution of functions in, a particular execution environment” [ISO/IEC 2005].

- A floating-point value with all bits zero represents a zero floating-point value. This assumption means that initializing memory with all-bits-zero (such as with `calloc`) initializes all floating-point objects to a zero value.
- A pointer-to-function can be converted to a pointer-to-void and back to a pointer-to-function without changing the value. This is true of all POSIX systems.
- Integers have a twos-complement representation. This assumption means that the bitwise operators produce well-defined results upon signed or unsigned integers, subject to restrictions upon the range of values produced.
- Integers are available for 8-, 16-, 32-, and 64-bit values. This assumption means that the library provides standardized type definitions for `int8_t`, `int16_t`, `int32_t`, and `int64_t`.

While not guaranteed by the C standard, these assumptions are frequently true for most implementations and allow for the development of smaller, faster, and less complex software. *The CERT C Secure Coding Standard* encourages the use of a static assertion to validate that these assumptions hold true for a given implementation (see guideline "DCL03-C. Use a static assertion to test the value of a constant expression") [Seacord 2008].

Because most code is constructed with these portability assumptions, it is generally counterproductive to diagnose code constructs that do not strictly conform. This would produce extensive diagnostic warnings in most code bases, and these flagged nonconformities would largely be perceived as false positives by developers who have made assumptions about the range of target platforms for the software.

Consequently, conformance testing for the *CERT C Secure Coding Standard* is performed with respect to one or more specific implementations. A certificate is generated for the product version, but each separate target implementation increases the cost of conformance testing. It is incumbent upon the developer requesting validation to provide the appropriate bindings for implementation-defined and unspecified behaviors evaluated during conformance testing.

1.7 SCALe for Energy Delivery Systems

Because of the flexibility of the C language, software developed for different application domains often has significantly different characteristics. For example, applications developed for the desktop may be significantly different than applications developed for embedded systems.

For example, one of the *CERT C Secure Coding Standard* rules is "ARR01-C. Do not apply the `sizeof` operator to a pointer when taking the size of an array." Applying the `sizeof` operator to an expression of pointer type can result in under allocation, partial initialization, partial copying, or other logical incompleteness or inconsistency if, as is usually the case, the programmer means to determine the size of an actual object. If the mistake occurs in an allocation, then subsequent operations on the under-allocated object may lead to buffer overflows. Violations of this rule are frequently, but not always, a coding error and software vulnerability. Table 1 illustrates the ratio of true positives (bugs) to flagged nonconformities in four open source packages.

Table 1: True Positives (TP) Versus Flagged Nonconformities (FNC)

Software System	TP/FNC	Ratio
Mozilla Firefox version 2.0	6/12	50%
Linux kernel version 2.6.15	10/126	8%
Wine version 0.9.55	37/126	29%
xc, version unknown	4/7	57%

The ratio of true positives to flagged nonconformities shows that this checker is inappropriately tuned for analysis of the Linux kernel, which has anomalous results. Customizing SCALe to work with energy system software will help eliminate false positives in the analysis of such code, decrease the time required to perform conformance testing, and subsequently decrease the associated costs.

2 Conformance Testing

This section describes the processes implemented in SCALe for conformance testing against a secure coding standard.

2.1 Conformance Testing Outcomes

Software systems can be evaluated against one or more secure coding standards. Portions of a software system implemented in languages for which a coding standard is defined and for which conformance tests are available can be evaluated for conformance to those standards. For example, a software system that is partially implemented in PL/SQL, C, and C# can be tested for conformance against *The CERT C Secure Coding Standard*. The certificate issued will identify the programming language composition of the system and note that the PL/SQL and C# components are not covered by the conformance test.

For each secure coding standard, the source code is found to be provably nonconforming, conforming, or provably conforming against each guideline in the standard as shown in Table 2.

Table 2: Conformance Testing Outcomes

Provably nonconforming	The code is provably nonconforming if one or more violations of a rule are discovered for which no deviation has been allowed.
Conforming	The code is conforming if no violations of a rule can be identified.
Provably conforming	The code is provably conforming if the code has been verified to adhere to the rule in all possible cases.

Strict adherence to all rules is unlikely, and, consequently, deviations associated with specific rule violations are necessary. Deviations can be used in cases where a true positive finding is uncontested as a rule violation, but the code is nonetheless determined to be secure. This may be the result of a design or architecture feature of the software or because the particular violation occurs for a valid reason that was unanticipated by the secure coding standard. In this respect, the deviation procedure allows for the possibility that secure coding rules are overly strict. Deviations will not be approved for reasons of performance, usability, or to achieve other nonsecurity attributes in the system. A software system that successfully passes conformance testing must not present known vulnerabilities resulting from coding errors.

Deviation requests are evaluated by the lead assessor, and if the developer can provide sufficient evidence that deviation will not result in a vulnerability, the deviation request will be accepted. Deviations should be used infrequently because it is almost always easier to fix a coding error than it is to provide an argument that the coding error does not result in vulnerability.

Once the evaluation process has been completed, CERT delivers to the client a report detailing the conformance or nonconformance of the code to the corresponding rules in the secure coding standard.

2.2 SCALe Laboratory Environment

Figure 3 shows the SCALe laboratory environment established at CERT.

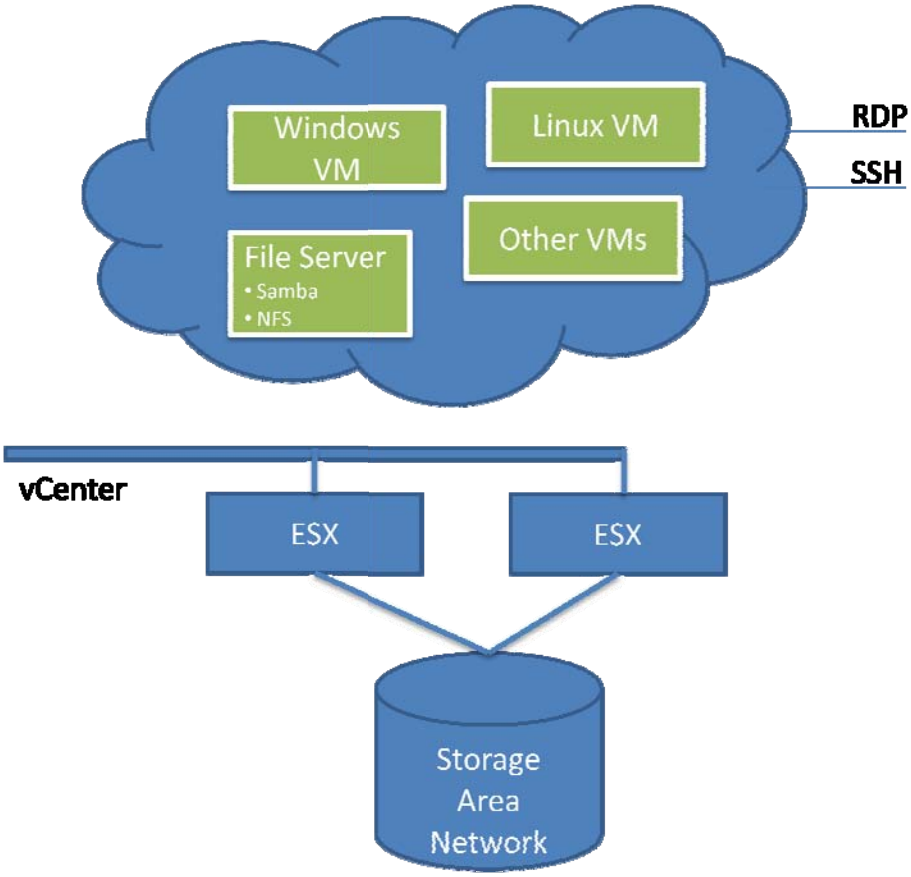

Figure 3: Source Code Analysis Laboratory

The SCALe laboratory environment consists of two servers running VMware ESX hypervisors. These are supported by a large storage area network (SAN) with redundant storage and backup capabilities. The two ESX servers support a collection of virtual machines (VMs) that can be configured to support analysis in various environments, such as Windows XP and Linux. A VMware vCenter Server provides control over the virtual environment.

The VMs are connected by a segmented-off network and to a file server running Samba and NFS. The Windows VMs can be remotely accessed from within the CERT network by using Remote Desktop Protocol (RDP) and the Linux VMs by using Secure Shell (SSH). The machines are otherwise disconnected from the internet.

Source code being analyzed is copied onto the file server, where it is available to all the analysis VMs. Analyzers and other tools are installed through a similar process or by using vCenter.

2.3 Conformance Testing Process

Figure 4 illustrates the SCALe conformance testing process. The client provides the software containing the code for analysis. This software must build properly in its build environment, such as Microsoft Windows/Visual Studio or Linux/GCC. It may produce compiler warnings but may not produce fatal errors. If the target operational environment is different than the build environment, the target environment must be fully specified, including all implementation-defined behaviors.

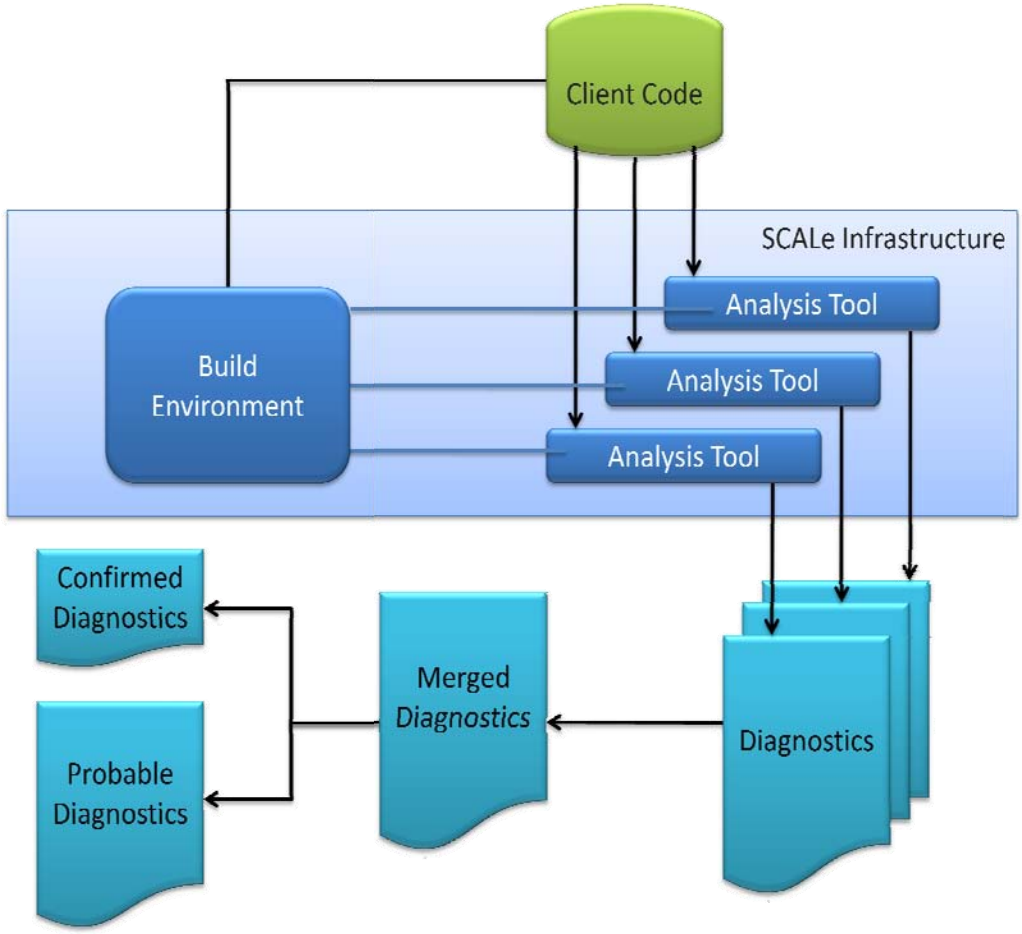

Figure 4: SCALe Conformance Testing Process

2.4 The Use of Analyzers in Conformance Testing

The client code is analyzed using multiple analyzers.[6] Section 1.5 contains additional background information on various types of analysis. Each analyzer accepts the client code as input and produces a set of flagged nonconformities.

Dynamic analysis tools must be able to run the program, which requires not only the correct execution environment but also suitable representative inputs. Additionally, the execution environment may include custom or special-purpose hardware, test rigs, and other equipment. These hurdles can make dynamic analysis tools challenging. The final report provided to the client documents the degree to which dynamic analysis was applied during conformance testing.

Also, an analyst may manually review the source code, using both structured and unstructured techniques, and record any violations of secure coding rules discovered. However, manual code scanning costs considerably more than automated analysis, and the results depend more on the skill and tenacity of the analyst.

[6] The C Secure Coding Rules Study Group defines an analyzer to be the mechanism that diagnoses coding flaws in software programs. This may include static analysis tools, tools within a compiler suite, and code reviewers.

Each analyzer produces a set of flagged nonconformities. Diagnostic formats vary with each tool, but they typically include the following information:

- name of source file where the flagged nonconformity occurs
- flagged nonconformity line number
- flagged nonconformity message (error description)

Some diagnostic messages may indicate a violation of a secure coding guideline or security violation, and others may not. Analyzer diagnostic warnings that represent violations of secure coding guidelines are mapped to the respective guideline, typically using a regular expression. This mapping can be performed directly by the tool or by the SCALe infrastructure. Analyzers that directly support a mapping to the CERT secure coding standards include Compass/ROSE, LDRA Testbed,[7] and Klocwork.[8]

When possible, SCALe also uses dynamic analysis and fuzz testing techniques to identify coding defects and for true/false positive analysis in addition to the routinely performed static analysis. An example of this is the basic fuzzing framework (BFF) developed by CERT. The BFF has two main parts:

- a Linux VM that has been optimized for fuzzing
- a set of scripts and a configuration file that orchestrate the fuzzing run

The VM is a stripped-down Debian installation with the following modifications:

- The Fluxbox window manager is used instead of the heavy Gnome or KDE desktop environments.
- Fluxbox is configured not to raise or focus new windows. This can help in situations where you may need to interact with the guest operating system (OS) while a graphical user interface (GUI) application is being fuzzed.
- Memory randomization is disabled for reproducibility.
- VMware Tools is installed, which allows the guest OS to share a directory with the host.
- The OS is configured to automatically log in and start X.
- The `sudo` command is configured not to prompt for a password.
- The `strip` command is symlinked to `/bin/true`, which prevents symbols from being removed when an application is built.

The goal of fuzzing is to generate malformed input that causes the target application to crash. The fuzzer used by the BFF is Sam Hocevar's zzuf application.[9] CERT chose zzuf for its deterministic behavior, number of features, and lightweight size. By invoking zzuf from a script (`zzuf.pl`), additional aspects of a fuzzing run are automatable:

- Collect program `stderr` output, Valgrind `memcheck`, and `gdb` backtrace. This information can help a developer determine the cause of a crash.

---

[7] http://www.ldra.com/certc.asp

[8] http://www.klocwork.com/solutions/security-coding-standards/

[9] http://caca.zoy.org/wiki/zzuf

- De-duplication of crashing test cases. Using `gdb` backtrace output, `zzuf.pl` determines if a crash has been encountered before. By default, duplicate crashes are discarded.
- Minimal test case generation. When a mutation causes a crash, the BFF will generate a test case where the number of bytes that are different from the seed file is minimized. By providing a minimal test case, the BFF simplifies the process of determining the cause of a crash.

The `zzuf.pl` reads the configuration options from the `zzuf.cfg` file. This file contains all of the parameters relevant to the current fuzz run, such as the target program and syntax, the seed file to be mutated, and how long the target application should be allowed to run per execution. The configuration file is copied to the guest OS when a fuzzing run has started. The zzuf script periodically saves its current progress within a fuzzing run as well. These two features work together to allow the fuzzing VM to be rebooted at any point, allowing the VM to resume fuzzing at the last stop point. The fuzzing script also periodically touches the `/tmp/fuzzing` file. A Linux software watchdog checks for the age of this file, and if it is older than the specified amount of time, the VM is automatically rebooted. Because some strange things can happen during a fuzzing run, this robustness is necessary for full automation. The `zzuf.pl` script takes this one step further by collecting additional information about the crashes. Cases that are determined to be unique are saved.

In addition to the BFF, CERT has developed a GNU Compiler Collection (GCC) prototype of the as-if infinitely ranged integer (AIR) model that, when combined with fuzz testing, can be used to discover integer overflow and truncation vulnerabilities. Assuming that the source code base can be compiled with an experimental version of the GCC 4.5.0 compiler, it may be possible to instrument the executable using AIR integers. AIR integers either produce a value equivalent to that obtained using infinitely ranged integers or cause a runtime-constraint violation. Instrumented fuzz testing of libraries that have been compiled using a prototype AIR integer compiler has been effective in discovering vulnerabilities in software and has low false positive and false negative rates [Dannenberg 2010].

With static tools, the entire code base is available for analysis. AIR integers, on the other hand, can only report constraint violations if a code path is taken during program execution and the input data causes a constraint violation to occur.

2.5 Conformance Test Results

CERT provides conformance test results to the client following step 5, “CERT analyzes source code and generates conformance test report,” as shown in the SCALe process overview in Figure 1, and again following step 7, “CERT issues conformance tests results and certificate.”

When available, violations that do not prevent successful conformance testing, or other diagnostic information, can be provided to the client for informational purposes.

2.5.1 Conformance Test Results Generation

The SCALe lead assessor integrates flagged nonconformities from multiple analyzers into a single diagnostic list. Flagged nonconformities that reference the same rule violation, file, and line number are grouped together and assigned to the same analysts based on the probability that these are multiple reports of the same error. In case these do refer to different errors, the individual reports

are maintained for independent analysis. However, it still makes sense to assign these as a group because the locality makes it easier to analyze them together.

Diagnostic warnings may sometimes identify errors not associated with any existing secure coding rule. This can occur for three reasons. First, it is possible that a diagnostic represents a vulnerability not addressed by any existing secure coding rule. This may represent a gap in the secure coding standard, which necessitates the addition of a new secure coding guideline. Second, a diagnostic may have no corresponding secure coding rule because the diagnostic does not represent a security flaw. Many analysis tools report portability or performance issues that are not considered to be secure coding rule violations. Third and finally, the diagnostic may be a false positive, that is, a diagnostic for which it is determined that the code does not violate a rule. False positives may arise through the normal operation of an analyzer, for example, because of the failure of a heuristic test. Alternatively, they may represent a defect in the analysis tool and consequently an opportunity to improve it. It is important to remember, however, that simultaneously avoiding both false positives and false negatives is generally impossible. Once a flagged nonconformity is determined to be a false positive, it is not considered or analyzed further.

Finally, the merged flagged nonconformities must be evaluated by a SCALe analyst to ascertain whether they are true or false positives. This is the most effort-intensive step in the SCALe process because there may be thousands of flagged nonconformities for a small- to medium-sized code base. Inspecting each flagged nonconformity is cost-prohibitive and unnecessary because it is possible to be confident—with a specified level of risk—that no true positives escape detection through statistical sampling and analysis.

Homogeneous buckets group flagged nonconformities based on the specific analyzer checker that reported it (as determined by examining the diagnostic). A statistical sampling approach selects a random sample of flagged nonconformities from a given bucket for further investigation. The specific statistical sampling approach used is called lot tolerance percent defective (LTPD) single sampling [Stephens 2001]. This LTPD reference uses an industry standard consumer risk of 10 percent, meaning that there is only a 10 percent chance of the security analyst being wrong in declaring a bucket of flagged nonconformities free of true positives based on the selected nominal limiting quality (defined in Table 3). The LTPD decision tables guiding the sample size for a given bucket require the following parameters as inputs:

1. bucket size—the number of flagged nonconformities for a given analyzer checker from which a sample will be investigated
2. nominal limiting quality (LQ)—the minimum percentage of true positives within a bucket of flagged nonconformities that the sampling plan will detect with 90 percent confidence and consequently confirm a violation of the coding rules

For the purposes of SCALe, the nominal LQ is assumed to be 2 percent. Note that the higher the LQ percentage, the smaller the sample of nonconformities for further investigation.

The above parameters, when used in conjunction with published LTPD tables, will determine the required sample size (n), from a bucket of flagged non-conformities associated with a given analyzer checker, that must be investigated by the SCALe analyst. Table 3 presents the set of the most likely scenarios that will be encountered by the security analysts, as derived from *The*

Handbook of Applied Acceptance Sampling [Stephens 2001].[10] The column headings contain the nominal LQ in percent, the row headings represent the bucket size, and their intersections in the table body are the sample size required by the nominal LQ and bucket size.

Table 3: Nominal Limiting Quality

Bucket Size (# of flagged nonconformities for a given analyzer checker)	Nominal Limiting Quality in Percent (LQ)					
	0.5%	0.8%	1.25%	2.0%	3.15%	5.0%
	Sample Size					
16 to 25	100% sampled	100% sampled	100% sampled	100% sampled	100% sampled	100% sampled
25 to 50	100% sampled	100% sampled	100% sampled	100% sampled	100% sampled	28[11]
51 to 90	100% sampled	100% sampled	100% sampled	50	44	34
91 to 150	100% sampled	100% sampled	90	80	55	38
151 to 280	100% sampled	170	130	95	65	42
281 to 500	280	220	155	105	80	50
501 to 1,200	380	255	170	125	125*	80*
1,201 to 3,200	430	280	200	200*	125*	125*
3,201 to 10,000	450	315	315*	200*	200*	200*

Assuming there are zero true positives found in a sample, the security analyst will be able to declare, for example, "Based on an investigation of a random sample of flagged nonconformities within a given bucket (of an analyzer checker), there is 90 percent confidence that the bucket of flagged nonconformities for a given analyzer checker contains no more than 2 percent true positives," where 2 percent true positives is the previously determined nominal LQ.

The procedure consists of the following steps for each bucket:

1. Identify the nominal LQ desired for the security analysis. For example, a 5 percent nominal LQ implies that the sampling scheme will identify buckets that have 5 percent or more true positives. The available tables offer LQ percentages of 0.5, 0.8, 1.25, 2.0, 3.15, 5.0 percent, and higher. The default LQ value for SCALe is 2 percent.
2. Identify the bucket size (number of flagged nonconformities within a bucket for a given analyzer checker).
3. Use the table to identify the required sample size (*n*). Note that at the 2 percent LQ, all flagged nonconformities are investigated if the bucket size totals 50 or fewer.
4. Randomly select the specified number (*n*) of flagged nonconformities from the bucket.

---

[10] For purposes of SCALe, the allowable number of defects found in a sample for a given quality level (Ac) is constrained to zero with the implication that any true positive found in a sample will be a basis for rejecting the bucket and declaring a violation of the security rule.

[11] If the required sample size is greater than the bucket size, then the sample size is the bucket size.

* At this LQ value and bucket size, the sampling plan would allow one observed true positive in the sample investigated, but the SCALe analyst would continue using the zero observed true positive rule to decide if the bucket is acceptable or not.

5. Investigate each flagged nonconformity in the sample to determine whether it is a false or true positive flagged nonconformity, and label it accordingly.
6. If all flagged nonconformities in the sample are false positives, all remaining flagged nonconformities in the bucket are discarded as false positives.
7. If a flagged nonconformity in the sample is determined to be a violation of the secure coding rule, it is categorized as a *confirmed violation*. No further investigation is conducted of the remaining nonconformities in the bucket, and these will continue to be categorized as *unknown*.

At the end of this process, there may be a small set of confirmed violations and a larger set of unknown or unevaluated violations. A confirmed violation represents a genuine security flaw in the software being tested and will result in the software being found provably nonconforming with respect to the secure coding guideline and failing to pass conformance testing. CERT will provide a list of unknown violations of the same secure coding rules to the client along with confirmed violations. The final diagnostic report consists of the confirmed violations together with the list of unknown violations.

2.5.2 Additional Documentation

Each rule provides additional information, including a description of the rule, noncompliant code examples, compliant solutions, and risk assessment, that provides software developers with an indication of the potential consequences of not addressing a particular vulnerability in their code (along with some indication of expected remediation costs). This metric is based on failure mode, effects, and criticality analysis (FMECA) [IEC 2006]. A development team can use this information to prioritize the repair of vulnerability classes.[12] It is generally assumed that new code will be developed to be compliant with all applicable guidelines.

As seen in Table 4, each rule in the *CERT C Secure Coding Standard* is scored on a scale of 1 to 3 for severity, likelihood, and remediation cost.

Table 4: Failure Mode, Effects, and Criticality Analysis

	Value	Meaning	Examples of Vulnerability	
Severity – How serious are the consequences of the rule being ignored?	1	low	denial-of-service attack, abnormal termination	
	2	medium	data integrity violation, unintentional information disclosure	
	3	high	run arbitrary code	
	Value	**Meaning**		
Likelihood – How likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability?	1	unlikely		
	2	probable		
	3	likely		
	Value	**Meaning**	**Detection**	**Correction**
Cost – How much will mitigating the vulnerability cost?	1	high	manual	manual
	2	medium	automatic	manual
	3	low	automatic	automatic

[12] Vulnerability metrics, such as the Common Vulnerability Scoring System (CVSS), measure the characteristics and impacts of specific IT vulnerabilities, not the risk from a coding rule violation.

2.6 Tracking Diagnostics Across Code Base Version

Infrequently, source code submitted for conformance assessment will be discovered to be free from secure coding violations on the initial assessment. More commonly, at least a single iteration is required. Consequently, this iteration has been designed into the process. Often, multiple iterations are required to discover and eliminate secure coding violations in software that has not been developed in conformance with the appropriate secure coding standards.

Depending on the analyzers used, it is not uncommon for code bases to have substantial numbers of false positives in addition to the true positives that caused the software to fail conformance testing. False positives must be eliminated before a software system can be determined to be conforming. However, analyzing the code to determine which diagnostics are false positives is time consuming and labor intensive. Furthermore, this process needs to be repeated each time the code base is submitted for analysis. Consequently, preventing the issuance of diagnostics determined to be false positives can reduce the cost and time required for conformance testing in most cases.

Diagnostics determined to be false positives can be eliminated in a variety of ways. Code constructs may be diagnosed because they correspond to common programmer errors. In other cases, these same code constructs may be intentional, but the analyzer cannot determine that a particular usage is secure. In these cases, the programmer simply needs a mechanism to express design intent more clearly.

Design intent can be expressed with the stylistic use of code or with special annotations. For example, given a guideline such as "FIO04-C. Detect and handle input and output errors," the following line of code would require a diagnostic:

```
puts("...");                // diagnostic required
```

However, the following code would be considered conforming:

```
if (EOF == puts("..."))     // okay: error handled
  exit(1);
```

If the failure to test the return value from the `puts` function was intentional, this design intent could be expressed by casting the resulting expression to `void`:

```
(void)puts("...");          // don't care about errors here
```

Special comments or pragmas may also be used for this purpose. For example, lint is silent about certain conditions if a special comment such as `/*VARARGS2*/` or `/*NOTREACHED*/` is embedded in the code pattern triggering them. The comment

```
/*NOTREACHED*/
```

is equivalent to

```
#pragma notreached
```

Of course, to suppress the diagnostic, both approaches must be recognized by the analyzer, and there is no standard set of stylistic coding conventions (although some conventions are more widely adopted than others).

Both approaches also require modification of source code, which, of course, is not a process or output of conformance testing. In fact, diagnostics are typically unsuppressed during analysis to ensure that secure coding violations are not inadvertently being suppressed.

A related approach is frequently referred to as "stand-off annotations," in which the annotations are external to the source code. This approach is more practical for SCALe and other processes in which the source code cannot be modified.

In step 6 of the SCALe process overview shown in Figure 1, the client has the opportunity to repair nonconforming code and can send the system back to CERT for a further assessment. Because the initial and subsequent code bases are separated by time and potentially multiple code restructurings, it can be difficult to match a new flagged nonconformity with a flagged nonconformity from an earlier version of the system. No matching technique will be perfect for all users, and it may fail in two ways:

1. It may fail to match a flagged nonconformity that should have been matched, so the false positive reappears.
2. It may erroneously match a flagged nonconformity that should have been treated separately. In this case the old flagged nonconformity's annotation will replace the newer flagged nonconformity. If the old flagged nonconformity was annotated as a false positive and the new flagged nonconformity is a true positive, then the user may never see it, creating a false negative.

GrammaTech CodeSonar, Coverity Prevent, and Fortify Source Code Analysis (SCA) each have a proprietary solution for solving this problem. SCALe could use these proprietary mechanisms to indicate at the individual tool level which diagnostics are false positives and should no longer be reported. This solution may be effective, but it requires direct access to the tool (as opposed to dealing strictly with aggregate results), and this approach is only feasible when the underlying tool provides the mechanism. Another drawback is that the false positive must be silenced by the conformance tester in each reporting analyzer.

Following the initial generation of a diagnostic report as described in Section 2.5.1, each diagnostic also has a validity status: true, probably true, unknown, probably false, or false. Each diagnostic starts in the unknown state. Any diagnostic that is manually inspected by an auditor becomes true or false. When the audit is complete, all other diagnostics will be probably true or probably false. This information needs to be transferred from the previous conformance test to minimize the amount of time spent reevaluating false positive findings.

2.6.1 Standard Approach

A potentially feasible approach to standardization is to specify a `#pragma` for analyzers to implement. With the `_Pragma` operator, the pragma name and number would not need to be the same across tools, although it would help if the pragma name and number mapped to equivalent functionalities such as those being produced by the WG14 C Secure Coding Rules Study Group. The following code illustrates a standard approach to using pragmas to suppress diagnostics:

```
#ifdef SA_TOOL_A
#  define DISABLE_FOO \
```

```
    _Pragma(push: tool_a_maybe_foo, disable: tool_a_maybe_foo)
#  define RESTORE_FOO _Pragma(pop: tool_a_maybe_foo)
#elif defined SA_TOOL_B
   #...

   void f() {
     DISABLE_FOO();
     /* do bad foo */
     RESTORE_FOO();
   }
```

Unfortunately, there are serious practical obstacles to portability when using pragmas.

The biggest problem with pragmas is that even though the language requires implementations to ignore unknown pragmas, they tend to be diagnosed by strict compilers. Compliers that do not do so make debugging incorrect uses of otherwise recognized pragmas difficult.

Another caveat about pragmas is that they have the effect of applying to whole statements rather than to expressions. Consider this example:

```
   void f(FILE *stream, int value) {
     char buf[20];
     #pragma ignore IO errors
     fwrite(buf, 1, sprintf("%i", value), stream);
   }
```

The pragma silences the diagnostics for both I/O functions on the next line, and it is impossible to make it silence just one and not the other. Of course, it is possible to rewrite this code so that the pragma would apply only to a single function call.

Developers submitting software for analysis are not required to silence unwanted diagnostics.

2.7 Quality Control

2.7.1 Personnel

2.7.1.1 Training

All SCALe lab personnel undergo basic security training and specialized training as required. Everyone, including those with client-facing roles, must have a computer science degree or equivalent education, specific training in the application of a particular secure coding standard, and training in conformance assessment using SCALe.

Currently, conformance assessment is being performed only with the *CERT C Secure Coding Standard*; therefore secure coding training required for personnel is one of the following:

- Software Engineering Institute (SEI) *Secure Coding in C and C++*[13]
- Carnegie Mellon University 15-392 Special Topic: Secure Programming[14]
- Carnegie Mellon University 14-735 Secure Software Engineering[15]
- An equivalent course determined by CERT

Following completion of training, a new SCALe employee undergoes an apprenticeship with a trained SCALe staff person. Upon successful completion of the apprenticeship—where success is determined by skill and capability, not by the passage of time—the new SCALe employee may work independently. However, the new employee, and all employees of SCALe, will continue to work under the transparency and audit controls described in this section.

All SCALe staff members undergo ethics training to ensure that SCALe conforms to the requirements of CERT, the SEI, and ISO/IEC 17000.

2.7.1.2 Roles

There are a number of defined roles within the SCALe lab.

- SCALe build specialist
  - Responsibilities: Installs the customer build environment on the SCALe lab machines
- SCALe analyst.
  - Responsibilities: Evaluates flagged nonconformities to determine if they represent violations of secure coding rules.
  - Additional training: Analysts must satisfactorily complete a formative evaluation assessment, as discussed in Section 2.7.2.
- SCALe lead assessor
  - Responsibilities: Organizes and supervises assessment activities, including supervising analyzers, tool selection, and drafting of reports.
  - Additional training: Has performed at least three assessments as a SCALe analyst.
- SCALe assessment administrator
  - Responsibilities: Develops and administers analyzer assessments.
- SCALe manager
  - Responsibilities: Handles business relationships, including contracting, communications, and quality assurance.

2.7.2 Quality Assurance Procedures

Every point where human judgment comes into play is an opportunity for SCALe to generate results that are not reproducible. This will be mitigated. Each judgment point will have a documented process for making that judgment. Personnel will be trained to faithfully apply the

[13] http://www.sei.cmu.edu/training/p63.cfm

[14] http://www-2.cs.cmu.edu/afs/cs/usr/cathyf/www/ugcoursedescriptions.htm

[15] http://www.ini.cmu.edu/degrees/psv_msit/course_list.html

processes. A system of review will be established, applied, and documented. The judgment will include at least the following.

Flagged nonconformity assessment: Much of the work of a conformity assessment is the human evaluation of the merged flagged nonconformities produced by the automated assessment tools. Different SCALe analysts each evaluate a subset of the flagged nonconformities. The intersection of those subsets is not the null set and is known only to the lead assessor. Consequently, SCALe analysts will perform audits of each other while simply doing their work. Any disagreement in results between SCALe analysts triggers a root cause assessment and corrective action.

Client qualification: Client qualification refers to the readiness of the client to engage the SCALe lab for analysis. The SCALe manager applies guidelines to determine if the potential client has the organizational maturity to provide software along with the build environment, respond to communications, maintain standards and procedures, and so forth. The tangible work products form an audit trail. CERT will conduct periodic review of the audit trail.

Tool selection: Because there is great inter-tool variation in flagging nonconformities, the selection of tools can have considerable impact on results. It is critical that SCALe lab conformance testing results be repeatable regardless of which particular person is selecting the tool set. The SCALe manager specifies, applies, and audits well-defined procedures for tool selection.

Conformance Test Completion: Because there will be far more flagged nonconformities than will be evaluated by SCALe analysts, the SCALe process applies statistical methods. Determining when enough flagged nonconformities have been evaluated is a well-defined process documented in Section 2.5.1.

Report generation: Final reports will be based on a template of predetermined parts, including, but not limited to, a description of software and build environment, the client's tolerance for missing nonconformities (typically less that 10 percent), tool selection, merged and evaluated flagged nonconformities, and stopping criterion. Both the SCALe lead assessor and the SCALe manager will sign off on each report. Each report will be reviewed by SEI communications for conformance with SEI standards.

2.7.2.1 Attribute Agreement Analysis

Attribute agreement analysis is a statistical method to determine the consistency of judgment within and between different SCALe analysts. Popular within the behavioral sciences, attribute agreement analysis remains a key method to determining agreement within and between raters, in this case SCALe analysts [von Eye 2006].

Simply, attribute agreement analysis constructs and implements a brief experiment in which the SCALe analysts participate in a short exercise of rendering judgment on a series of flagged nonconformities. The exercise specifically includes a variety of flagged nonconformities mapped to different rules. Attribute agreement analysis evaluates the judgments as correct or incorrect, based on the flagged nonconformity being a true positive or a false positive. In these situations, an attribute agreement measures the true-or-false positive judgment similarly to the traditional use of attribute agreement analysis in the quality control domain for pass/fail situations. The attribute agreement measure provides feedback in several dimensions:

- individual accuracy (for example, what percentage of judgments are correct)

- individual consistency (for example, how consistent is the individual in rendering the same judgment across time for the same or virtually the same flagged nonconformity; often referred to as *repeatability*)
- group accuracy (for example, what percentage of the time does a specific group of SCALe analysts render the correct judgment)
- group consistency (for example, what percentage of the time does a specific group of SCALe analysts render the same judgment for a given flagged nonconformity across time; often referred to as *reproducibility*)

Any modern statistical package can easily determine the attribute agreement measures, which are interpreted as follows [Landis 1977]:

- Less than 0 (no agreement)
- 0–0.20 (slight agreement)
- 0.21–0.40 (fair agreement)
- 0.41–0.60 (moderate agreement)
- 0.61–0.80 (substantial agreement)
- 0.81–1 (almost perfect agreement)

A need may arise to assess both accuracy and consistency of SCALe analysts' judgments with measures that extend beyond the binary situation (correct or incorrect) to situations in which a judgment is a gradual measure of closeness to the right answer. In this case, analysts should use an alternative attribute agreement measure and interpret the output quite similarly to the Kappa coefficient, with results possible on the dimensions listed above. As such, Kendall coefficients serve well for judgments on an ordinal scale, in which incorrect answers are closer or farther away from the correct answer. A hypothetical example of a judgment on an ordinal scale would be if a SCALe analyst were asked to render judgment of the severity of a flagged nonconformity, say on a 10-point scale. If the true severity is, for example, 8, and two SCALe analysts provided answers of 1 and 7, respectively, then a severity judgment of 7 would have a much higher Kendall coefficient than the severity judgment of 1.

In conclusion, attribute agreement analysis may be conducted via small exercises with SCALe analysts rendering judgments on a reasonably-sized list of different types of flagged nonconformities mapped to the set of rules within the scope of a given code conformance test.

2.7.2.2 Formative Evaluation Assessment Using Attribute Agreement Analysis

SCALe analysts participate in a formative evaluation assessment as part of their training and certification. Certification of a candidate as a SCALe analyst requires attribute agreement scores of 80 percent or higher. In addition, acceptable thresholds for accuracy may be imposed separately for each rule.

The formative evaluation assessment implements a simple attribute agreement analysis as follows.

First, the SCALe assessment administrator identifies a preliminary set of 20 to 35 different flagged nonconformities (from the diagnostic output of a software system or code base) for the evaluation assessment. The administrator ensures that the preliminary set includes a variety of

diagnostic codes mapped to a representative collection of security rules. The administrator then identifies a second set of 20 to 35 different flagged nonconformities, such that there is a similarity mapping between each flagged nonconformity in the first set to a corresponding flagged nonconformity in the second set. The resulting complete set of 40 to 70 flagged nonconformities is then randomized and used as a test instrument for the participating SCALe analysts to evaluate.

Second, different SCALe analysts are identified to participate in the evaluation assessment. Initially, there must be at least two analysts to conduct the evaluation assessment. Subsequently, additional analysts will take the same evaluation assessment using the same or a similar set of flagged nonconformities.

Third, the SCALe analysts independently evaluate each flagged nonconformity within the complete set as either a true or false positive, recognizing true positives as rule violations.

Because human judgment can vary across time (for example, SCALe analysts may fall out of practice in exercising their judgment of flagged nonconformities) and because the scope and nature of flagged nonconformities and rules may vary across time, CERT retests SCALe analysts using attribute agreement analysis every three years as part of recertification.

Lastly, the SCALe manager uses the results of ongoing attribute agreement exercises to identify ways to improve the training of SCALe analysts, including possible additional job aids. SCALe analysts will also be interviewed for context information surrounding incorrect judgments as part of this improvement activity.

Thresholds will be maintained at established levels until and unless experience indicates that they should change.

2.7.2.3 Attribute Agreement Analysis Test Experiment

To qualify potential analyst candidates, the SCALe assessment administrator conducted an attribute agreement analysis test. The test consisted of 60 flagged nonconformities divided into pairs of similar flagged nonconformities, each having the same validity.

The administrator assigned all flagged nonconformities a numeric ID to identify pairs. The administrator constructed the test (and answer key) and assigned the test to four SCALe analyst candidates. The analyst candidates had no qualifications for code analysis other than being competent programmers. Each analyst candidate made a true or a false positive determination for each flagged nonconformity. Afterward, the analyst candidates and administrator compared results. While the administrator had initially created the answer key, the group came to different conclusions about some of the diagnostics. Table 5 presents the results of the test. The column marked "AA" contains the results for the assessment administrator's answer key, while the columns marked "AC #" are the results for the four analyst candidates tested.

Table 5: Attribute Agreement Analysis Test Results

ID	Rule	Group	AA	AC 1	AC 2	AC 3	AC 4
1	DCL32-C	False	False	True	False	False	False
1	DCL32-C	False	False	False	True	False	False
2	OOP32-CPP	True	True	True	True	True	True
2	OOP32-CPP	True	True	True	True	True	True
3	MEM41-CPP	False	False	True	True	False	False
3	MEM41-CPP	False	False	True	True	False	False
4	MEM40-CPP	False	False	False	True	False	False
4	MEM40-CPP	False	False	True	True	True	False
5	EXP34-C	False	False	True	False	True	False
5	EXP34-C	False	False	True	False	True	False
6	DCL35-C	True	False	True	False	False	True
6	DCL35-C	True	False	True	True	False	True
7	ERR33-CPP	False	False	False	True	False	False
7	ERR33-CPP	False	False	False	True	True	False
8	ERR33-CPP	True	True	False	False	False	True
8	ERR33-CPP	True	True	False	True	True	True
9	EXP36-C	True	False	False	True	False	True
9	EXP36-C	True	False	False	True	True	True
10	EXP35-CPP	True	True	True	True	True	True
10	EXP35-CPP	True	True	True	True	False	True
11	DCL36-C	False	False	True	True	True	False
11	DCL36-C	False	False	True	True	True	False
12	FLP36-C	False	False	False	False	False	False
12	FLP36-C	False	False	False	False	False	False
13	FIO30-C	False	False	False	False	False	False
13	FIO30-C	False	False	False	True	True	False
14	FLP34-C	False	False	True	False	False	False
14	FLP34-C	False	False	True	False	False	False
15	FLP34-C	True	True	True	True	True	True
15	FLP34-C	True	True	True	False	True	True
16	ARR30-C	False	True	False	True	True	False
16	ARR30-C	False	True	False	True	True	False
17	STR38-C	True	True	True	True	True	True
17	STR38-C	True	True	True	True	True	True
18	OOP37-CPP	True	True	True	True	True	True
18	OOP37-CPP	True	True	False	True	False	True
19	OOP37-CPP	False	False	True	True	False	False
19	OOP37-CPP	False	False	True	True	False	False
20	DCL31-C	False	False	False	False	True	False
20	DCL31-C	False	False	False	False	True	False
21	DCL31-C	False	False	False	False	False	False

ID	Rule	Group	AA	AC 1	AC 2	AC 3	AC 4
21	DCL31-C	False	False	False	False	False	False
22	INT31-C	False	False	False	False	False	False
22	INT31-C	False	False	True	False	False	False
23	INT31-C	False	False	False	False	False	False
23	INT31-C	False	False	False	False	False	False
24	INT31-C	False	False	True	True	False	False
24	INT31-C	False	False	True	True	False	False
25	MSC34-C	True	True	True	True	True	True
25	MSC34-C	True	True	True	True	True	True
26	MSC34-C	False	False	True	True	False	False
26	MSC34-C	False	False	True	True	False	False
27	EXP36-C	False	False	True	True	False	False
27	EXP36-C	False	False	True	True	True	False
28	INT35-C	True	True	True	True	True	True
28	INT35-C	True	True	True	True	True	True
29	EXP34-C	True	True	False	False	True	True
29	EXP34-C	True	True	True	True	True	True
30	MEM41-CPP	False	False	True	True	False	False
30	MEM41-CPP	False	False	False	False	False	False

The administrator's findings correlated strongly with the group results. The administrator shared 54 of 60 answers with the group, for a score of 90 percent. The analyst candidates' scores showed considerably lower correlation, with results of 56.7, 58.3, 70, and 75 percent, respectively. This would rate analyst candidates 1 and 2 in moderate agreement with the group and analyst candidates 3 and 4 in substantial agreement.

Most analyst candidates displayed only moderate consistency. They gave the same answer to the similar flagged nonconformity pairs most of the time but not always. Analyst candidate 1 gave the same answer to similar flagged nonconformity pairs 24 out of 30 times, for a consistency score of 83.3 percent. The second and third analyst candidates gave the same answer 23 out of 30 times, for a consistency score of 76.7 percent. The fourth analyst candidate was extremely consistent, giving the same answer 29 out of 30 times for a consistency score of 96.7 percent.

Attribute agreement analysis can also be conducted with the Minitab 16 Statistical Software[16] as shown in Figure 5. The "Within Appraisers" section depicts the degree of internal consistency for each analyst and the administrator. Only the administrator and analyst candidate 4 have acceptable Kappa values indicating very good internal consistency. All of the *p* values are less than 0.05, indicating that these results are statistically significant and not due to chance. The "Each Appraiser vs Standard" section depicts how accurate each analyst and the administrator are in getting the correct answer. Again, the administrator and analyst candidate 4 have acceptable Kappa values, indicating very good accuracy in determining both false and true positives. The *p* values less than 0.05 for the administrator and analyst candidate 4 indicate that the Kappa values are statistically significant and not due to chance.

[16] http://www.minitab.com/en-US/products/minitab/default.aspx

Attribute Agreement Analysis for Response

Within Appraisers

Assessment Agreement

Appraiser	# Inspected	# Matched	Percent	95% CI
Administrator	30	30	100.00	(90.50, 100.00)
Analyst1	30	24	80.00	(61.43, 92.29)
Analyst2	30	23	76.67	(57.72, 90.07)
Analyst3	30	22	73.33	(54.11, 87.72)
Analyst4	30	30	100.00	(90.50, 100.00)

Matched: Appraiser agrees with him/herself across trials.

Fleiss' Kappa Statistics

Appraiser	Response	Kappa	SE Kappa	Z	P(vs > 0)
Administrator	False	1.00000	0.182574	5.47723	0.0000
	True	1.00000	0.182574	5.47723	0.0000
Analyst1	False	0.58333	0.182574	3.19505	0.0007
	True	0.58333	0.182574	3.19505	0.0007
Analyst2	False	0.48718	0.182574	2.66839	0.0038
	True	0.48718	0.182574	2.66839	0.0038
Analyst3	False	0.46429	0.182574	2.54300	0.0055
	True	0.46429	0.182574	2.54300	0.0055
Analyst4	False	1.00000	0.182574	5.47723	0.0000
	True	1.00000	0.182574	5.47723	0.0000

Each Appraiser vs Standard

Assessment Agreement

Appraiser	# Inspected	# Matched	Percent	95% CI
Administrator	30	27	90.00	(73.47, 97.89)
Analyst1	30	14	46.67	(28.34, 65.67)
Analyst2	30	14	46.67	(28.34, 65.67)
Analyst3	30	17	56.67	(37.43, 74.54)
Analyst4	30	30	100.00	(90.50, 100.00)

Matched: Appraiser's assessment across trials agrees with the known standard.

Fleiss' Kappa Statistics

Appraiser	Response	Kappa	SE Kappa	Z	P(vs > 0)
Administrator	False	0.78022	0.129099	6.04356	0.0000
	True	0.78022	0.129099	6.04356	0.0000
Analyst1	False	0.13237	0.129099	1.02533	0.1526
	True	0.13237	0.129099	1.02533	0.1526
Analyst2	False	0.16440	0.129099	1.27343	0.1014
	True	0.16440	0.129099	1.27343	0.1014
Analyst3	False	0.38286	0.129099	2.96563	0.0015
	True	0.38286	0.129099	2.96563	0.0015
Analyst4	False	1.00000	0.129099	7.74597	0.0000
	True	1.00000	0.129099	7.74597	0.0000

Figure 5: Attribute Agreement Analysis using Minitab

The analyst candidate errors resulted from a number of factors. Many candidates, while knowledgeable in C, possessed only a rudimentary knowledge of C++. The analyst candidates expressed a

lack of confidence with C++. However, when ignoring the results for C++ specific rules, there were 21 flagged nonconformity pairs, or 42 individual flagged nonconformities. For the C subset, the administrator shared 36 of 42 answers with the group, for a score of 85.7 percent. The analyst candidates scored 59.5, 40.5, and 69 percent, respectively. We would conclude from this that while lack of C++ experience made the analyst candidates less confident with their test results, they did not do significantly worse on the C++-flagged nonconformities than they did with the C-flagged nonconformities.

Some errors resulted from a lack of in-depth knowledge of C. For example, two analyst candidates incorrectly confused the harmless format string "%d\n" with the more notorious format string "%n". Ignorance of the Windows function calls and types employed by the code led to some mistakes.

Analyst candidates also had difficulty deciding if a diagnostic was an actual violation of a CERT rule, even when they fully understood the code. Two analyst candidates incorrectly marked diagnostic pair 26 as false because the diagnostic text referred to a MISRA rule that had been violated, and the analyst candidates had not considered that MISRA was not authoritative for the purpose of this test [MISRA 2004].

Some diagnostics were incorrectly marked true because they indicated portability problems rather than security problems. For instance, diagnostic pair 24 indicated code that was not portable across different platforms but was perfectly secure when run on its intended platform. The code depended on specific integer sizes, which are guaranteed by particular implementations of C, but not by the C standard.

There were also many errors caused by insufficient whole-program analysis. Interestingly, all of the cases where the analyst candidate disagreed with the group arose because the administrator failed to perform sufficient whole-program analysis. One (or more) of the analyst candidates performed a more comprehensive analysis on a diagnostic, causing them to come to a different conclusion and convince the group that the administrator's answer key was incorrect.

These scores lead us to conclude that analyst candidates without special training are not qualified to produce accurate or consistent analysis results. This may be because of the analyst candidates' lack of knowledge or experience or because of poor testing conditions. Furthermore the test should be specified more rigorously so that analyst candidates are not unduly influenced by external authorities, such as MISRA.

Rules that require whole-program analysis are also problematic because whole-program analysis is prohibitively expensive, and analysis costs scale exponentially with program size. Many rules try to not require whole-program analysis, but some cannot be enforced without it. For instance, checking for memory leaks requires detailed knowledge of the entire codebase. Evaluating these rules only in the context of a particular function can result in false positives being identified as actual violations. In many cases, the developer may need to provide the evidence that these are not true violations.

3 Conformance Testing

3.1 Introduction

In general, objective third-party evaluation of a product provides confidence and assurance that the product conforms to a specific standard. The CERT SCALe assesses a software system, determines if it conforms to a CERT secure coding standard, and provides evidence to that effect. The services are performed under a service agreement.

Conformance testing by a recognized and respected organization such as CERT ensures the impartiality of the assessment, ensures fair and valid testing processes, and fosters confidence and acceptance of the software by consumers in the public and private sectors.

According to the results of a recent survey conducted for the Independent Association of Accredited Registrars (IAAR), the main motives organizations cited for obtaining a third-party certification of conformance to a quality standard were "customer mandate" (29 percent), "competitive pressure or advantage" (17 percent), "continuous improvement based on customer requirements" (16 percent), and "improve quality" (14 percent). Less frequently cited were "implementation and control of best practice" (10 percent) and "corporate mandate" (9 percent). "Reduce cost," "risk management," and "legal reasons" were each cited by 1 percent of respondents [ANAB 2008].

For many organizations, product certification yields financial benefits because of cost reduction and new sources of revenue. Among respondents to the IAAR survey, 86 percent of companies certified in quality management realized a positive return on investment (ROI). An ROI of more than 10 percent was reported by 26 percent of respondents to the survey.

While undergoing third-party audits to become certified may be voluntary, for many organizations there are compelling reasons to do so:

- improve the efficiency and effectiveness of operations
- satisfy customer requirement
- satisfy contractual, regulatory, or market requirement
- instill organizational discipline
- demonstrate to shareholders, regulators, and the public that a software product has been audited
- instill customer confidence
- identify issues that may be overlooked by those inside the organization, providing fresh internal improvement strategies

Common elements of conformance assessment include impartiality, confidentiality, complaints and appeals, and information disclosure policy.

3.1.1 Impartiality

CERT resides within Carnegie Mellon University's Software Engineering Institute, a federally funded research and development center. The SEI and CERT are frequently called upon to provide impartial third-party assessments.

3.1.2 Complaints and Appeals

CERT records and investigates complaints received from customers or other parties and, when warranted, takes corrective action. CERT monitors the results to ensure the effectiveness of corrective actions.

It is not uncommon for a software developer to dispute a finding as being a false positive. In these cases, the software developer is required to provide evidence to CERT that the finding is a false positive. CERT then reviews this evidence and either corrects the finding or refutes the evidence. In cases where the coding construction is determined to be a violation of a secure coding rule but can be demonstrated to present no vulnerability because of architectural, design, or deployment constraints, the developer may request, and will be granted, a deviation.

3.1.3 Information Disclosure Policy

CERT holds proprietary information (such as source code) in the strictest confidence and maintains its confidentiality by using at least as much care as the client uses to maintain the confidentiality of its own valuable proprietary and confidential information. CERT will not disclose this information to employees other than to those whose official duties require the analysis of the source code. CERT will not disclose proprietary information to any third party without the prior written consent of the customer. All obligations of confidentiality survive the completion of the conformance assessment process.

CERT may publish company-specific information in aggregate form and without attribution to source.

3.2 CERT SCALe Seal

Developers of software that has been determined by CERT as conforming to a secure coding standard may use the seal shown in Figure 6 to describe the conforming software on the developer's website. The seal must be specifically tied to the software passing conformance testing and not applied to untested products, the company, or the organization.

Figure 6: CERT SCALe Seal

Except for patches that meet the criteria below, any modification of software after it is designated as conforming voids the conformance designation. Until such software is retested and determined to be conforming, the new software cannot be associated with the CERT SCALe seal.

Patches that meet all three of the following criteria do not void the conformance designation:

- The patch is necessary to fix a vulnerability in the code or is necessary for the maintenance of the software.
- The patch does not introduce new features or functionality.
- The patch does not introduce a violation of any of the rules in the secure coding standard to which the software has been determined to conform.

Use of the CERT SCALe seal is contingent upon the organization entering into a service agreement with Carnegie Mellon University and upon the software being designated by CERT as conforming.

3.3 CERT SCALe Service Agreement

Organizations seeking SCALe conformance testing will abide by the SCALe policies and procedures required by the SCALe Service Agreement. Organizations submitting software code for conformance testing will follow these basic processes:

1. A service agreement must be fully executed by the organization and Carnegie Mellon University's Software Engineering Institute before conformance testing begins.
2. CERT evaluates the source code of the software against the identified CERT secure coding standard(s), specified in the statement of work, using the identified tools and procedures and provides an initial conformance test report to the client that catalogues all rule violations found as a result of the SCALe evaluation.
3. From receipt of the initial conformance test report, the client has 180 days to repair nonconforming code and/or prepare documentation that supports the conclusion that identified violations do not present known vulnerabilities and resubmit the software and any deviation requests for a final evaluation of the software against the specified CERT secure coding standard(s).

4. CERT will evaluate any deviation requests and reevaluate the software against the specified CERT secure coding standard(s) and provide a final conformance test report to the client.
5. Clients are permitted to use the CERT SCALe seal on their website in connection with successful product conformance testing after the product version has passed the applicable conformance test suite(s). Clients may describe the product version as having been determined by CERT to conform to the CERT secure coding standard.
6. Clients whose software passes the conformance testing agree to have their product version listed on the CERT web registry of conforming systems.

3.3.1 Conformance Certificates

SCALe validation certificates include the client organization's name, product name, product version, and registration date. Certificates also include a list of applicable guidelines and an indication if, for a particular guideline, the source code being tested was determined to be provably conforming or conforming.

Register of Conforming Products

CERT will maintain an online certificates registry of systems that conform to CERT secure coding standards at https://www.securecoding.cert.org/registry.

3.4 SCALe Accreditation

CERT will not initially seek American National Standards Institute (ANSI), International Organization for Standardization (ISO), or NIST accreditation for SCALe from an accreditation agency. However, CERT will endeavor to implement processes, procedures, and systems that comply with national and international standards. As needed, the program can submit for accreditation by the following agencies:

- ISO/IEC. This agency has published *ISO/IEC 65*, which provides principles and requirements for the competence, consistency, and impartiality of third-party certification bodies evaluating and certifying products (including services) and processes. This standard is under revision and is scheduled to be released as *17065* in July of 2011. The agency has also published *ISO/IEC 17025:2005 General Requirements for the Competence of Testing and Calibration Laboratories*, which specifies the requirements for sound management and technical competence for the type of tests and calibrations SCALe undertakes. Testing and calibration laboratories that comply with *ISO/IEC 17025* also operate in accordance with *ISO 9001*.
- NIST National Voluntary Laboratory Accreditation Program (NVLAP). NVLAP provides third-party accreditation to testing and calibration laboratories. NVLAP's accreditation programs are established in response to Congressional mandates, administrative actions by the federal government, and requests from private-sector organizations and government agencies.
- NVLAP operates an accreditation system that is compliant with *ISO/IEC 17011:2004 Conformity assessment*. It provides general requirements for bodies accrediting conformance assessment bodies, which requires that the competence of applicant laboratories be assessed by the accreditation body against all of the requirements of *ISO/IEC 17025: 2005 General requirements for the competence of testing and calibration laboratories*.

3.5 Transition

Transition of SCALe to practice will follow the SEI's transition strategy to grow the concept through engagement with external organizations or SEI partners via a series of deliberate steps. The proof-of-concept phase will occur with a piloting program of SCALe that engages a small number of clients. During this phase, CERT will test and refine processes, procedures, systems, and outputs.

After the pilot phase, CERT will engage a small number of additional organizations that will be licensed to sponsor SCALe laboratories within themselves. Each organization will be licensed to perform the assessment, issue the conformance assessment report, report results to CERT, and be subject to annual quality audits of all processes, procedures, hardware, and software.

3.6 Conformance Test Results

As of the publication of this report, CERT has completed the analysis of one energy delivery system and begun analyzing a second.

3.6.1 Energy Delivery System A

Table 6 shows the flagged nonconformities reported from analysis of the first energy delivery system. The analysis was performed using four static analysis tools supplemented by manual code inspection. Dynamic analysis was not used.

Table 6: Flagged Nonconformities, Energy Delivery System A

	Manual	Analyzer A	Analyzer B	Analyzer C	Analyzer D	Total
DCL31-C			0	705		705
DCL32-C			0	119	1	120
DCL35-C			0	51		51
DCL36-C			0	19		19
EXP30-C			0	3		3
EXP34-C			0	54	4	58
EXP36-C			0	24	6	30
EXP37-C			0	49		49
INT31-C		1	4	3,588		3,593
INT35-C		6	0			6
FLP34-C			0	9		9
FLP36-C			0	1		1
ARR30-C		4	2	1		7
STR31-C			3			3
STR36-C			0	6		6
STR37-C			0		1	1
STR38-C			7			7
MEM34-C			0		2	2
FIO30-C			0	12		12
ENV30-C			0		3	3
SIG30-C	1		0			1
CON33-C	1		0			1
MSC31-C			0		1	1
MSC34-C			0	587		587
Total	2	11	16	5,228	18	5,275

The first column marked “Manual” shows violations that were discovered through manual code inspection, while the four columns marked “Analyzer A,” “Analyzer B,” “Analyzer C,” and “Analyzer D” show the number of flagged nonconformities detected by each of the four analysis tools used in this analysis.

Table 7 shows the results of analysis of the flagged nonconformities by the SCALe analysts and SCALe lead assessor combined.

Table 7: Analysis Results, Energy Delivery System A

	False	True	Unknown	Total
DCL31-C	705			705
DCL32-C	2	2	116	120
DCL35-C		2	49	51
DCL36-C	19			19
EXP30-C	2	1		3
EXP34-C	2	4	52	58
EXP36-C	4	4	22	30
EXP37-C	4	1	44	49
INT31-C	1,999		1,594	3,593
INT35-C		6		6
FLP34-C	7	2		9
FLP36-C	1			1
ARR30-C	7			7
STR31-C	3			3
STR36-C	4	2		6
STR37-C		1		1
STR38-C	3	4		7
MEM34-C	2			2
FIO30-C	12			12
ENV30-C	3			3
SIG30-C		1		1
CON33-C		1		1
MSC31-C		1		1
MSC34-C	6	2	579	587
Total:	2,785	34	2,456	5,275

The “False” and “True” columns document the number of flagged nonconformities that were determined to be false and true positives, respectively. Normally it is sufficient to stop after finding one true positive, but in cases with a small number of flagged nonconformities, all the results were evaluated to collect data about the true positive and flagged nonconformity rates for the analyzer checkers. Flagged nonconformities that were not evaluated are marked as “Unknown.”

This particular energy control system violated at least 15 of the CERT C secure coding rules. In nine other cases, manual analysis eliminated possible rule violations as false positives.

3.6.2 Energy Delivery System B

The second energy delivery system was also evaluated by four static analysis tools supplemented by manual inspection. Two of the tools (analyzers A and B) were also used in the analysis of energy delivery system A. The other two analyzers were used for the first time in the analysis of energy delivery system B. Table 8 shows the flagged nonconformities found from the analysis of the second energy delivery system.

Table 8: Flagged Nonconformities, Energy Delivery System B

	Manual	Analyzer B	Analyzer C	Analyzer E	Analyzer F	Total
ARR30-C			17	1		18
ARR36-C	2		106			108
DCL35-C			47			47
DCL36-C			2			2
EXP30-C			2			2
EXP33-C			308	1	25	334
EXP34-C		21	483	68	49	621
EXP36-C			109			109
EXP37-C			40			40
EXP40-C			20			20
FIO30-C			2	6		8
FLP34-C			324			324
FLP35-C			9			9
INT31-C		8	7,562	5	2	7,577
INT32-C				7	2	9
MEM30-C				1	2	3
MEM31-C		10	7		4	21
MEM33-C			4			4
MEM34-C			1			1
MSC34-C			362			362
PRE30-C			4			4
PRE31-C			11			11
STR30-C			11			11
STR31-C		1		5	44	50
STR32-C		1		10		11
STR33-C		1				1
Total	2	42	9,431	104	128	9,707

Table 9 shows the results of analysis of the flagged nonconformities by the SCALe analysts and SCALe lead assessor combined. Unfortunately, this analysis was not completed. Where all flagged nonconformities for a rule were unknown, the nonconformities have not been evaluated.

Table 9: Analysis Results, Energy Delivery System B

	False	Suspicious	True	Unknown	Total
ARR30-C				18	18
ARR36-C			2	106	108
DCL35-C				47	47
DCL36-C			2		2
EXP30-C	2				2
EXP33-C	5			329	334
EXP34-C	13		3	605	621
EXP37-C				40	40
EXP40-C				20	20
FIO30-C	3	2	3		8
FLP35-C				9	9
INT31-C	603		2	6,971	7,576
INT32-C				9	9
MEM30-C	3				3
MEM31-C				21	21
MEM33-C			4		4
MEM34-C			1		1
MSC34-C	326		36		362
PRE30-C	4				4
PRE31-C				11	11
STR30-C				11	11
STR31-C			1	50	51
STR32-C				11	11
STR33-C			1		1

Based on our experience with analyzing energy delivery system A, we added a new category of "suspicious." This category includes flagged nonconformities that could not easily be proven to be either true or false positives. This was frequently the case for dereferencing null pointers, for example, where the pointer dereferences were unguarded but it was difficult to prove that the pointer was never null without performing whole-program analysis. Suspicious violations are treated as false positives in that they will not result in a system failing conformance testing and will not stop the analyst from analyzing other flagged nonconformities reported against the same coding rule. These are reported as suspicious so that the developer can examine these flagged nonconformities and take appropriate measures.

Overall, energy delivery system B had considerably more flagged nonconformities than energy delivery system A, a significant number of which have already been determined to be true positives.

4 Related Efforts

This section describes related conformance assessment activities in today's marketplace.

4.1 Veracode

Veracode's[17] Risk Adjusted Verification Methodology allows organizations developing or procuring software to measure, compare, and reduce risks related to application security. Veracode uses static binary analysis, dynamic analysis, and manual penetration testing to identify security flaws in software applications. The basis for the VERAFIED security mark is the Security Quality Score (SQS). SQS aggregates the severities of all security flaws found during the assessment and normalizes the results to a scale of 0 to 100. The score generated by each type of assessment is then mapped to the application's business criticality (assurance level), and those applications that reach the highest rating earn the VERAFIED security mark.

4.2 ISCA Labs

ICSA Labs,[18] an independent division of Verizon Business, has been providing independent, third-party product assurance for end users and enterprises for 20 years. ICSA Labs says they provide "vendor-neutral testing and certification for hundreds of security products and solutions for many of the world's top security product developers and service providers" [Cybertrust 2010]. ICSA Labs provides services in three areas:

- Consortium Operations, Security Product Testing, and Certification Programs
- Custom Testing Services
- Accredited Government Testing Services

ICSA Labs is *ISO 17025:2005* accredited and *ISO 9001:2008* registered.

4.3 SAIC Accreditation and Certification Services

SAIC (Science Applications International Corporation)[19] provides security content automation protocol (SCAP) testing and monitoring of systems for security issues such as software deficiencies, configuration issues, and other vulnerabilities. The testing helps ensure that a computer's configuration is within guidelines set by the Federal Desktop Core Configuration. Notably, they became an accreditation body under the NIST accreditation to perform SCAP.

4.4 The Open Group Product Certification Services

The Open Group[20] has developed and operates an industry-based product certification program in several areas, including UNIX, CORBA, POSIX, and LDAP. They have developed and currently

[17] http://www.veracode.com/

[18] http://www.icsalabs.com/

[19] http://www.saic.com/infosec/testing-accreditation/scap.html

[20] http://www.opengroup.org/consortia_services/certification.htm

maintain conformance test suites for multiple technologies, including those listed above, the X Window System, Motif, Digital Video Broadcasting Multimedia Home Platform, Secure Electronic Transactions (SET), Common Data Security Architecture (CDSA), and Linux [Open Group 2010].

The Open Group product certification program provides formal recognition of a product's conformance to an industry standard specification. This allows suppliers to make and substantiate clear claims of conformance to a standard and allows buyers to specify and successfully procure conforming products that interoperate [Open Group 2010].

The Open Group's product certification programs are based on a supplier's claim of conformance; testing provides an indicator of conformance. Suppliers typically use test suites to establish confidence that their product conforms. To achieve certification, the supplier must provide a warranty of conformance ensuring the following [Open Group 2010]:

- products conform to an industry standard specification
- products remain conformant throughout their lifetimes
- the product will be fixed in a timely manner if there is a nonconformance

The Open Group acts as the independent certification authority for industry-based certification programs. As the certification authority, their web-based conformance testing system is tailored to guide suppliers through the process of certifying a product [Open Group 2010].

5 Future Work and Summary

5.1 Future Work

Work is continuing on the development of secure coding standards for C++, Java, and other programming languages. As these standards are completed and adequate tooling becomes available, SCALe will be extended to support conformance testing against these secure coding standards.

CERT will also expand SCALe's operational capability, including integrating additional commercial and research analyzers into the SCALe laboratory environment. This process includes acquiring tools, creating a mapping between diagnostics generated by the tool and CERT secure coding standards, and automating the processing of these diagnostics.

In addition to the use of acceptance sampling plans based on the lot tolerance percent defective, other techniques can be researched for use when greater amounts of data from conformance testing are available. These techniques, including Bayesian methods, may enable even more informed decisions for the stopping rules related to the investigation of flagged nonconformities for false positives. It is anticipated that such analysis will eventually be granular down to the flagged nonconformity and help further reduce the sample size of flagged nonconformities to be investigated.

Additionally, a number of techniques can be explored to characterize the performance of each of the security checker tools in terms of each tool's

- proportion of false positives to true positives
- ability to find certain classes of true positives that are not discovered by other analyzers

Given this information, more informed decisions can be made within each security analysis event in terms of which checker tools to employ. The SCALe lead assessor would discontinue the use of specific checkers that have a high proportion of false positives and little, if any, contribution to the identification of true positives above and beyond what the other checker tools are capable of finding.

5.2 Summary

Growing numbers of vulnerability reports and reports of software exploitations demand that underlying issues of poor software quality and security be addressed. Conformance with CERT secure coding standards is a measure of software security and quality that provides an indication of product security. SCALe provides a defined, repeatable process for conformance testing of software systems. Conformance testing against the CERT secure coding standard should help establish a market for secure software by allowing vendors to market software quality and security and also enable consumers to identify and purchase conforming products.

Bibliography

URLs are valid as of the publication date of this document.

[ANAB 2008]
ANSI-ASQ National Accreditation Board (ANAB). *The Third-Party Process: Waste of Resources or Added Value?* ANAB, 2008.
http://www.anab.org/media/9593/valueofaccreditedcertification.pdf

[Ashling Microsystems]
Ashling Microsystems. *Ashling Microsystems*. http://www.ashling.com/technicalarticles/APB201-TestbedDescription.pdf, 2010.

[CERT 2010a]
CERT. *The CERT C Secure Coding Standard, Version 2.0*. Software Engineering Institute, Carnegie Mellon University, 2010. https://www.securecoding.cert.org/confluence/x/HQE

[CERT 2010b]
CERT. *The CERT C++ Secure Coding Standard*. Software Engineering Institute, Carnegie Mellon University, 2010. https://www.securecoding.cert.org/confluence/x/fQI

[CERT 2010c]
CERT. *The CERT Oracle Secure Coding Standard for Java*. Software Engineering Institute, Carnegie Mellon University, 2010. https://www.securecoding.cert.org/confluence/x/Ux

[Christy 2007]
Christey, Steve & Martin, Robert A. *Vulnerability Type Distributions in CVE, Version: 1.1*. MITRE, 2007. http://cve.mitre.org/docs/vuln-trends/vuln-trends.pdf

[Cybertrust 2010]
Cybertrust. *About ISCA Labs*. http://www.icsalabs.com/about-icsa-labs, 2010.

[Dannenberg 2010]
Dannenberg, Roger B.; Dormann, Will; Keaton, David; Seacord, Robert C.; Svoboda, David; Volkovitsky, Alex; Wilson, Timothy; & Plum, Thomas. “As-If Infinitely Ranged Integer Model,” 91-100. *Proceedings 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE 2010)*, San Jose, CA, Nov. 2010. IEEE 2010.

[Heffley 2004]
Heffley, J. & Meunier, P. “Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?” *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS’04) - Track 9 - Volume 9*. Island of Hawaii, January 2004. IEEE Computer Society, 2004.

[IEC 2006]
International Electrotechnical Commission (IEC). *Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA), 2nd ed.* (IEC 60812). IEC, January 2006.

[IEEE Std 610.12 1990]
IEEE. *IEEE Standard Glossary of Software Engineering Terminology* (Std. 610.12-1990). IEEE, 1990.

[ISO/IEC 2004]
International Organization for Standardization, International Electrotechnical Commission (ISO/IEC). *ISO/IEC 17000:2004 Conformity Assessment — Vocabulary and General Principles, 1st edition.* ISO, 2004.

[ISO/IEC 2005]
International Organization for Standardization, International Electrotechnical Commission (ISO/IEC). *ISO/IEC 9899:1999 Programming Languages—C.* ISO, 2005. ISO http://www.open-std.org/JTC1/SC22/wg14/www/docs/n1124.pdf

[ISO/IEC 2007]
International Organization for Standardization, International Electrotechnical Commission (ISO/IEC). *ISO/IEC TR 24731-1:2007 Extensions to the C Library — Part I: Bounds-checking interfaces*. ISO, August 2007.

[ISO/IEC 2010a]
International Organization for Standardization, International Electrotechnical Commission (ISO/IEC). *ISO/IEC TR 24731-2:2010 Extensions to the C Library — Part II: Dynamic Allocation Functions.* ISO, 2010.

[ISO/IEC 2010b]
International Organization for Standardization, International Electrotechnical Commission (ISO/IEC). *Programming Languages—C++, Final Committee Draft,* ISO/IEC JTC1 SC22 WG21 N3092. ISO, March 2010.

[Jones 2010]
Jones, Larry. *WG14 N1539 Committee Draft ISO/IEC 9899:201x*. International Standards Organization, 2010.

[Landis 1977]
Landis, J. R. & Koch, G. G. “The Measurement of Observer Agreement for Categorical Data.” *Biometrics 33* (1977): 159-174.

[Landwehr 2008]
Landwehr, C. *IARPA STONESOUP Proposers Day*. IARPA, 2008. http://www.iarpa.gov/Stonesoup_Proposer_Day_Brief.pdf

[MISRA 2004]
Motor Industry Software Reliability Association (MISRA). *MISRA-C:2004: Guidelines for the Use of the C Language in Critical Systems*. MIRA, 2004.

[Morrow forthcoming]
Morrow, Tim; Seacord, Robert; Bergey, John; Miller, Phillip. *Supporting the Use of CERT Secure Coding Standards in DoD Acquisitions* (CMU/SEI-2010-TN-021). Software Engineering Institute, Carnegie Mellon University, forthcoming.

[Okun 2009]
Okun, V.; Gaucher, R.; & Black, P. E., eds. *Static Analysis Tool Exposition (SATE) 2008* (NIST Special Publication 500-279). NIST, 2009.

[Open Group 2010]
The Open Group. *Collaboration Services*.
http://www.opengroup.org/consortia_services/certification.htm, 2010.

[Plum 2005]
Plum, Thomas & Keaton, David M. "Eliminating Buffer Overflows, Using the Compiler or a Standalone Tool," 75-81. *Proceedings of the Workshop on Software Security Assurance Tools, Techniques, and Metrics.* Long Beach, CA, November 2005. U.S. National Institute of Standards and Technology (NIST), 2005.

[Plum 2009]
Plum, Thomas & Seacord, Robert C. *ISO/IEC JTC 1/SC 22/WG14/N1350 – Analyzability*. International Standards Organization (ISO), 2009.
http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1350.htm

[Saltzer 1975]
Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems." *Proceedings of the IEEE 63*, 9 (September 1975): 1278-1308.

[Seacord 2003]
Seacord, R. C.; Plakosh, D.; & Lewis, G. A. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[Seacord 2005]
Seacord, R. C. *Secure Coding in C and C++* (SEI Series in Software Engineering). Addison-Wesley Professional, 2005.

[Seacord 2008]
Seacord, R. *The CERT C Secure Coding Standard*. Addison-Wesley Professional, 2008.

[Stephens 2001]
Stephens, Kenneth S. *The Handbook of Applied Acceptance Sampling*. ASQ Quality Press, 2001.

[Takanen 2008]
Takanen, A.; DeMott, J.; & Miller, C. *Fuzzing for Software Security Testing and Quality Assurance*. 1. Artech House, Inc., 2008.

[von Eye 2006]
von Eye, Alexander & Mun, Eun Young. *Analyzing Rater Agreement: Manifest Variable Methods*. Lawrence Erlbaum Associates, 2006.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Source Code Analysis Laboratory (SCALe) for Energy Delivery Systems	FA8721-05-C-0003

6. AUTHOR(S)

Robert C. Seacord, William Dormann, James McCurley, Philip Miller, Robert Stoddard, David Svoboda, Jefferson Welch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TR-021

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2010-021

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Source Code Analysis Laboratory (SCALe) is an operational capability that tests software applications for conformance to one of the CERT® secure coding standards. CERT secure coding standards provide a detailed enumeration of coding errors that have resulted in vulnerabilities for commonly used software development languages. The SCALe team at CERT, a program of Carnegie Mellon University's Software Engineering Institute, analyzes a developer's source code and provides a detailed report of findings to guide the code's repair. After the developer has addressed these findings and the SCALe team determines that the product version conforms to the standard, CERT issues the developer a certificate and lists the system in a registry of conforming systems. This report details the SCALe process and provides an analysis of energy delivery systems. Though SCALe can be used in various capacities, it is particularly significant for conformance testing of energy delivery systems because of their critical importance.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software security, conformance testing, secure coding standards	57

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL