Source Code Analysis Laboratory (SCALe)

About CERT

Software Engineering Institute (SEI)

- Federally funded research and development center at Carnegie Mellon University

CERT Coordination Center (CERT/CC)

- Incident response
  - Morris worm (1988)

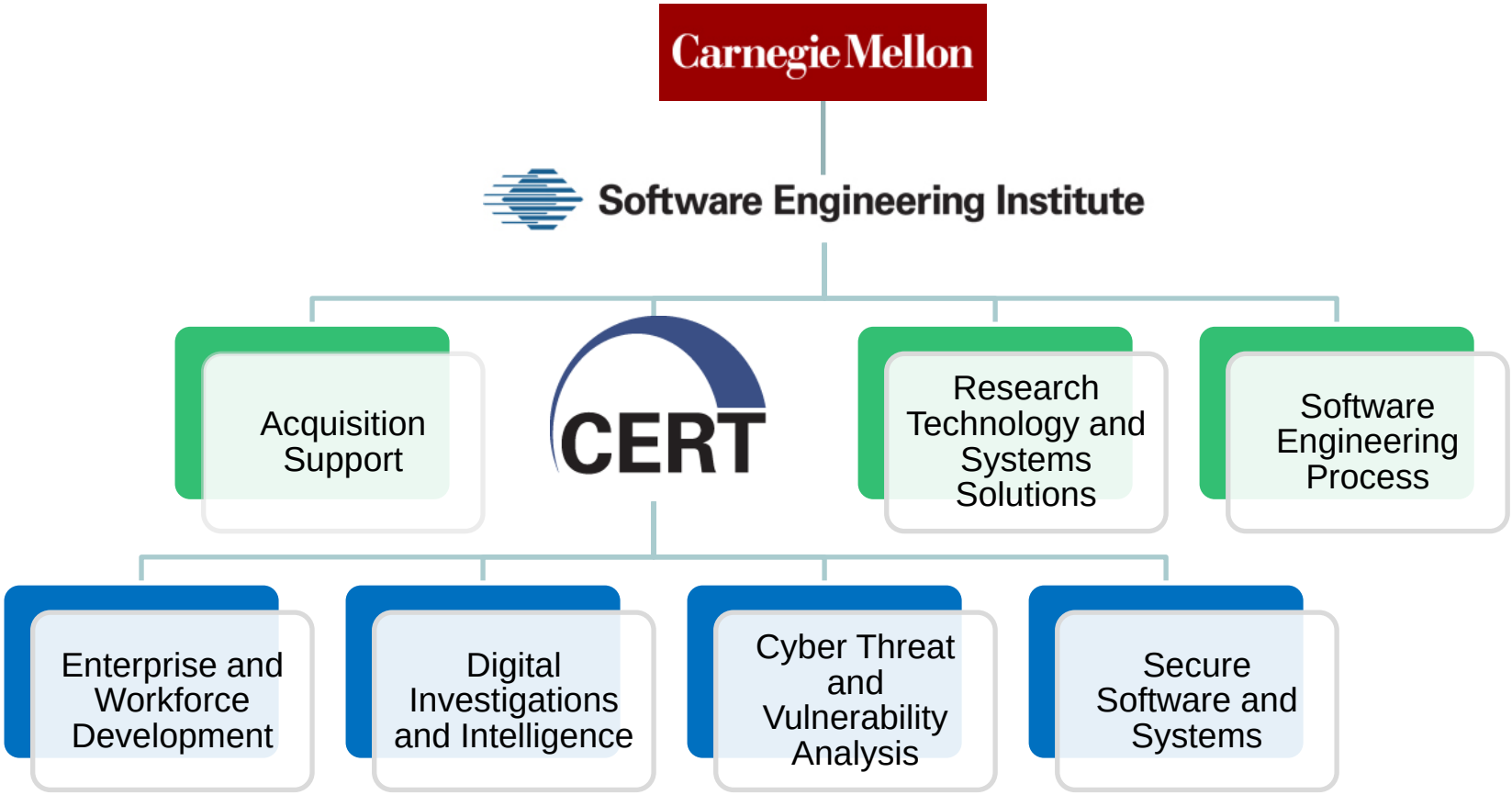
Cyber Threat and Vulnerability Analysis (CTVA)

- Vulnerability analysis and coordination
- Malicious code analysis
- Network analysis
- Incident handling

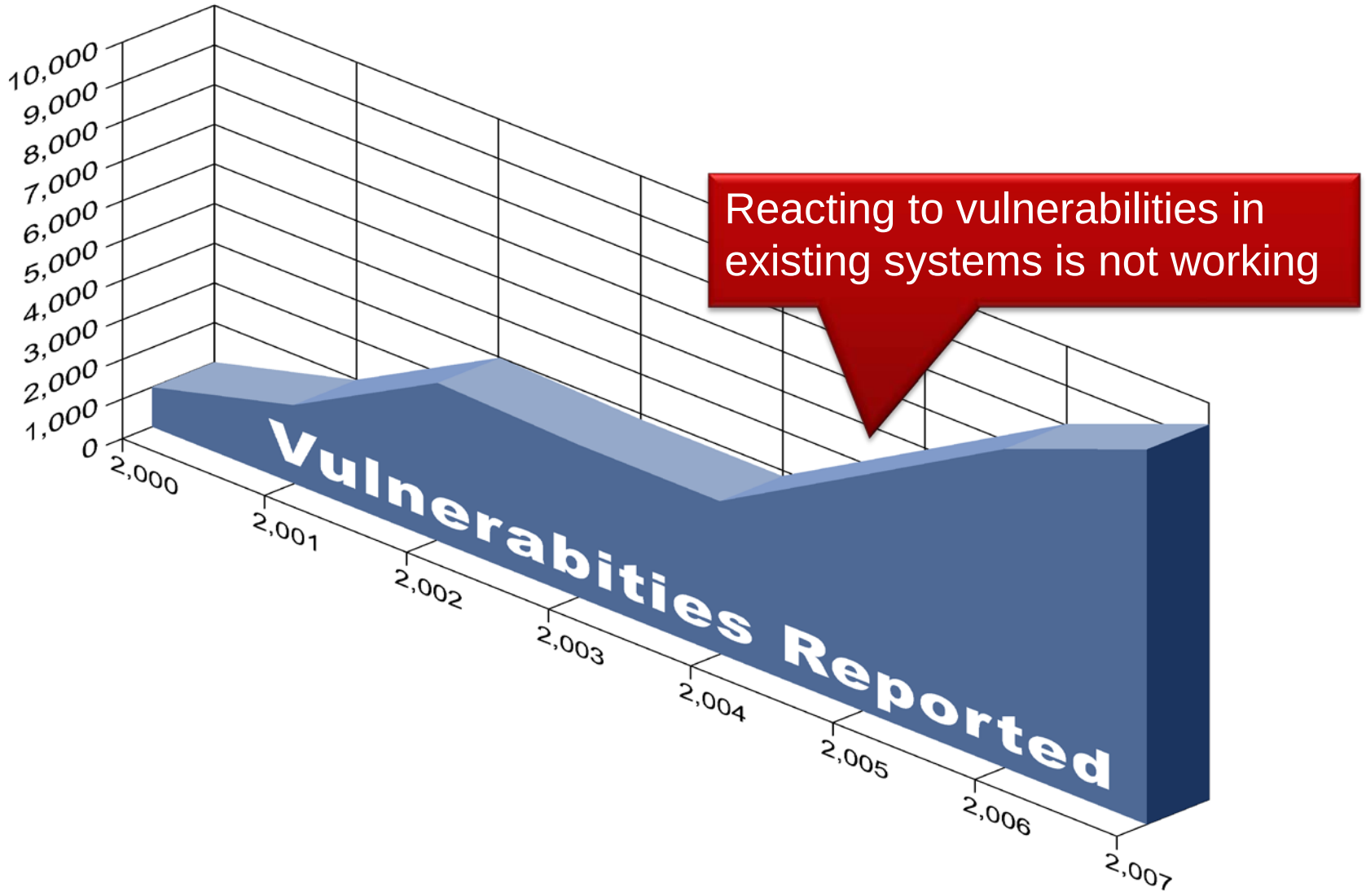
Secure Software and Systems (SSS)

- Secure Coding
- Cyber security engineering

CERT Program



Increasing Vulnerabilities



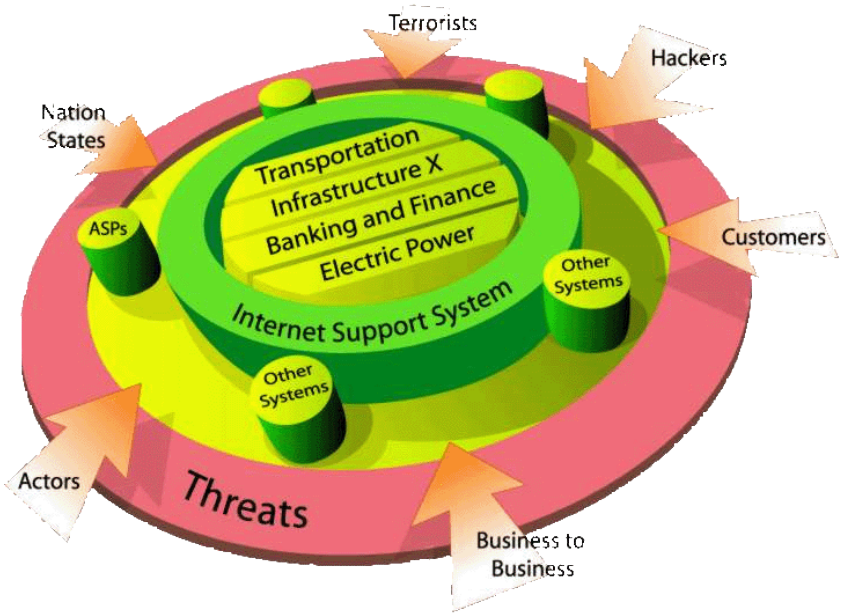
Why Software Security?

Developed nations' economies and defense depend, in large part, on the reliable execution of software

Software is ubiquitous, affecting all aspects of our personal and professional lives.

Software vulnerabilities are equally ubiquitous, jeopardizing:

- Personal identities
- Intellectual property
- Consumer trust
- Business services, operations, & continuity
- Critical infrastructures & government



Application Security

Secure Coding

Licensed to:
- Computer Associates
- Siemens

Open & free online course
- USC, Matt Bishop
- Stevens, Sven Dietrich
- CMU

SEI Secure Coding Course

University courses
- CMU
- Stevens Institute
- Purdue
- University of Florida
- Santa Clara University
- St. John Fisher College

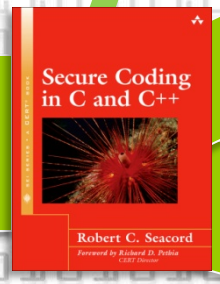
Secure Design Patterns

Influence International Standard Bodies

ISO IEC

WG14 C Secure Coding Rules Study Group

Analyzer conformance test



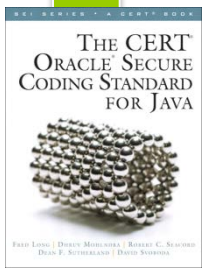
Adoption by Analyzer Tools
- LDRA
- Klocwork

SCALe Testbed

•Thread role analysis
• Security-enhanced compiler

CERT CONFORMANCE TESTED

C++



Adoption by software developers and acquirers
- Cisco
- Raytheon
- NAVSEA
- Lockheed Martin Aeronautics
- General Atomics
- Qualcomm

Breadth of impact

2003

Time

2014

CERT Secure Coding Standards

CERT C Secure Coding Standard

- Version 1.0 (C99) - published
- Version 2.0 (C11) - under development

CERT C++ Secure Coding Standard

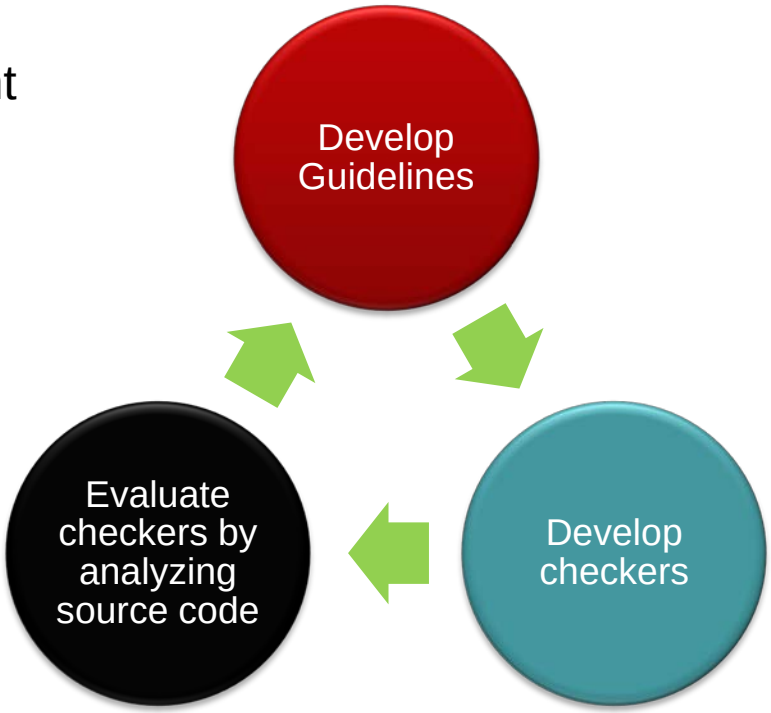
- Version 1.0 (C++ 11) under development

CERT Oracle Secure Coding Standard for Java

- Version 1.0 for Java SE 6 published
- Static analysis under development

The CERT Perl Secure Coding Standard

- Version 1.0 under development

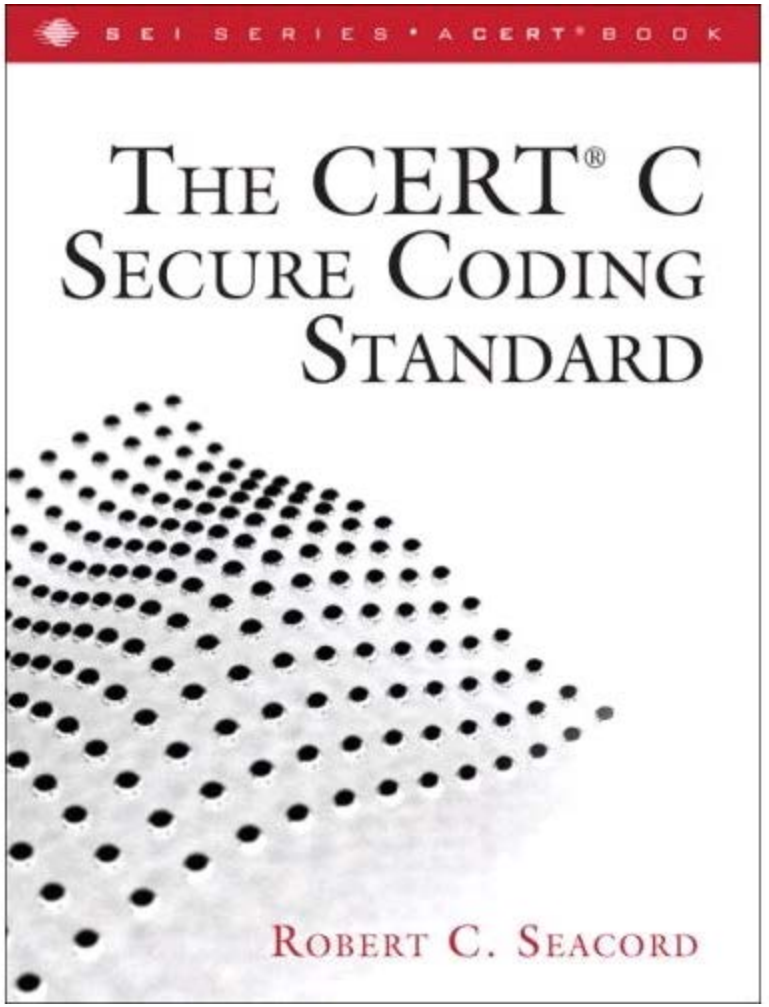


The CERT C Secure Coding Standard

Developed with community involvement, including over 500 registered participants on the wiki.

Version 1.0 published by Addison-Wesley in September, 2008.

- 134 Recommendations
- 89 Rules



Noncompliant Examples & Compliant Solutions

Noncompliant Code Example

In this noncompliant code example, the **`char`** pointer **`p`** is initialized to the address of a string literal. Attempting to modify the string literal results in undefined behavior.

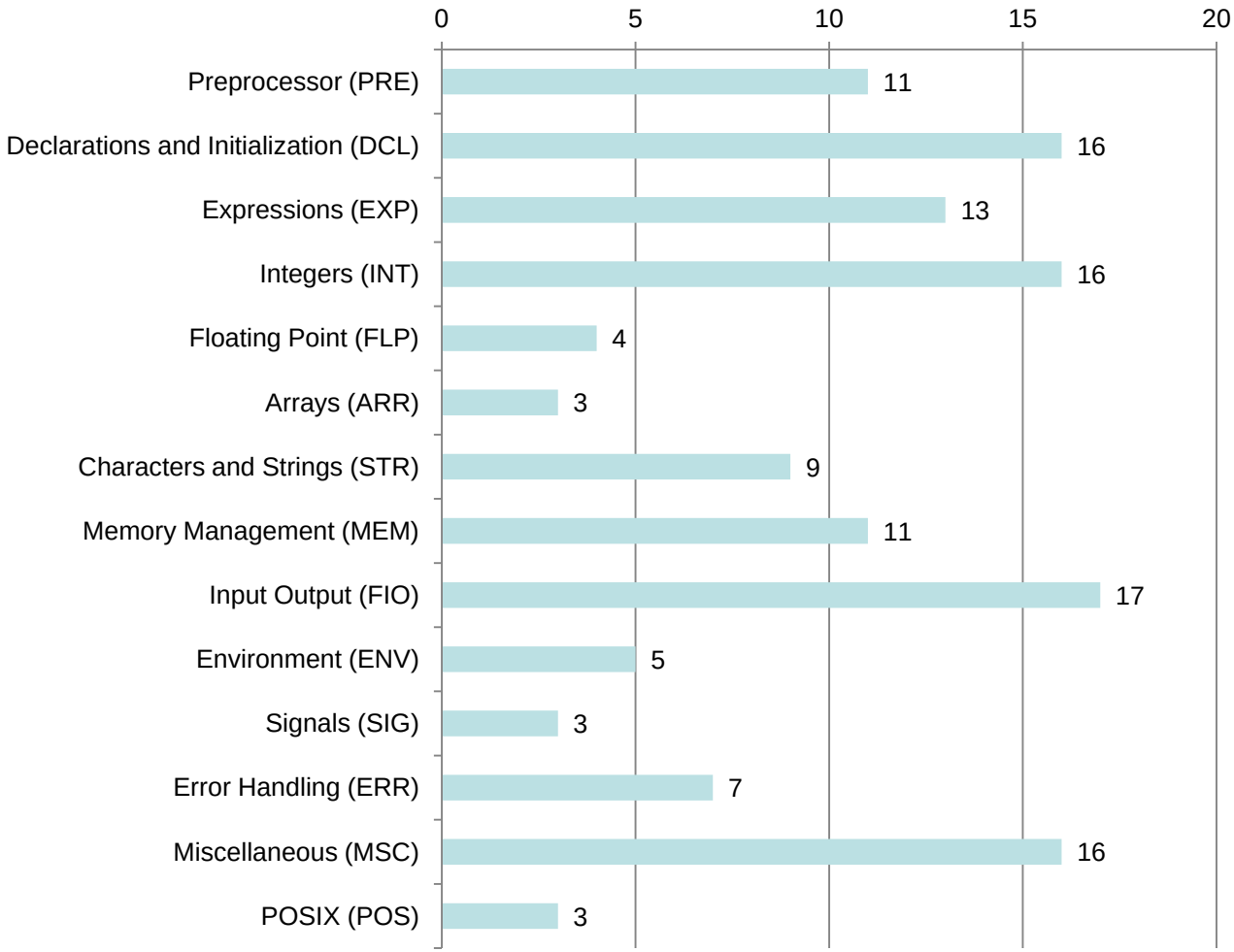
```
char *p = "string literal"; p[0] = 'S';
```

Compliant Solution

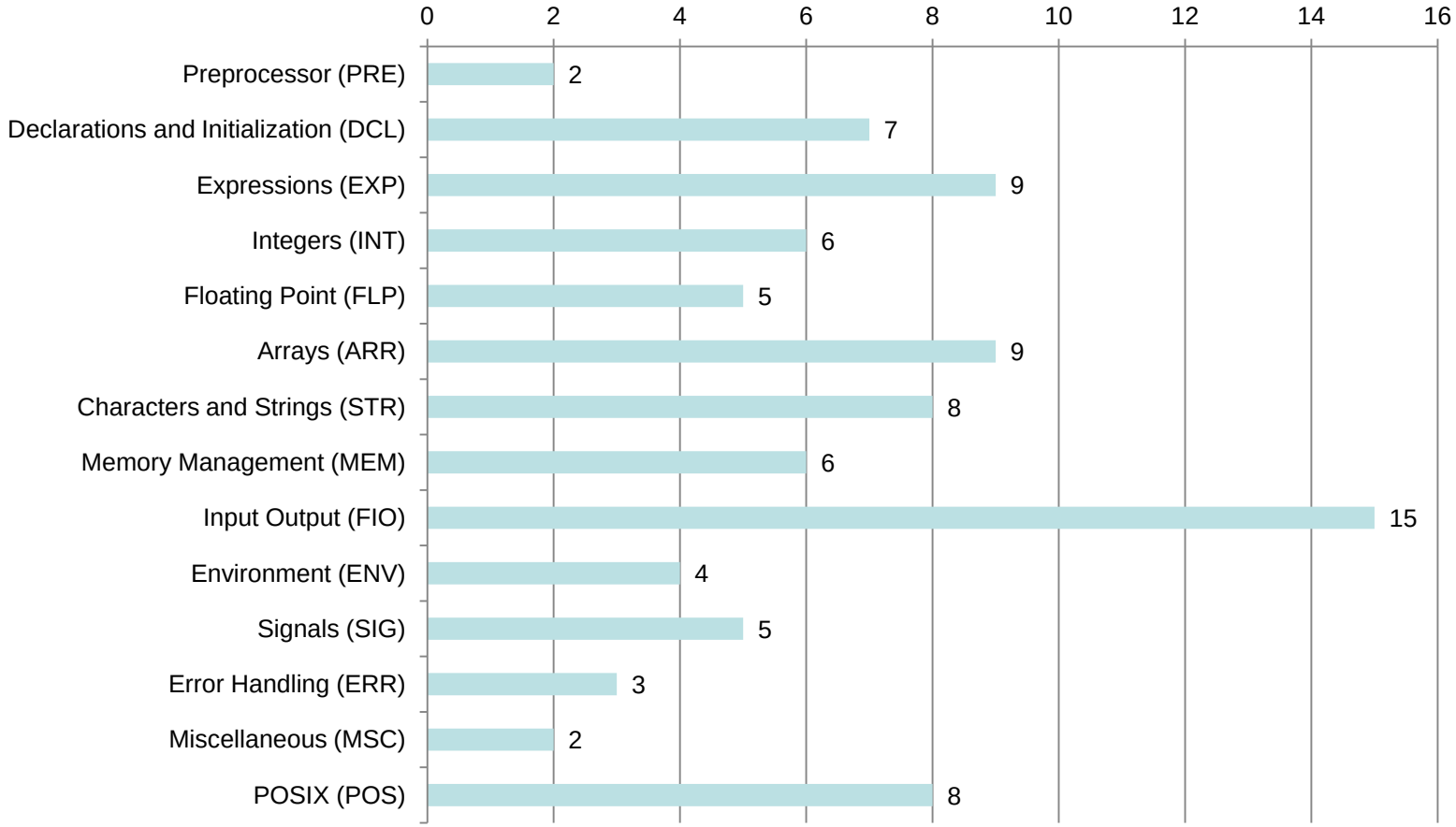
As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal in the space allocated to the character array **`a`**. The string stored in a can be safely modified.

```
char a[] = "string literal"; a[0] = 'S';
```

Distribution of C Recommendations



Distribution of C Rules



POSIX

Many of the core guidelines demonstrate compliant solutions that rely for POSIX-compliant systems.

The CERT C Secure Coding Standard also contains an appendix with guidelines (3 recommendations and 8 rules) for using functions that are defined as part of the POSIX family of standards but are not included in ISO/IEC 9899-1999.

These rules and recommendations are not part of the core standard because they do not apply in all C language applications and because they represent an incomplete set.

The intent of providing these guidelines is to demonstrate how rules and recommendations for other standards or specific implementations may be integrated with the core C99 recommendations.

Risk Assessment

Risk assessment is performed using failure mode, effects, and criticality analysis

Severity – how serious are the consequences of the rule being ignored?

Value	Meaning	Examples of Vulnerability
1	low	denial-of-service attack, abnormal termination
2	medium	data integrity violation, unintentional information disclosure
3	high	run arbitrary code

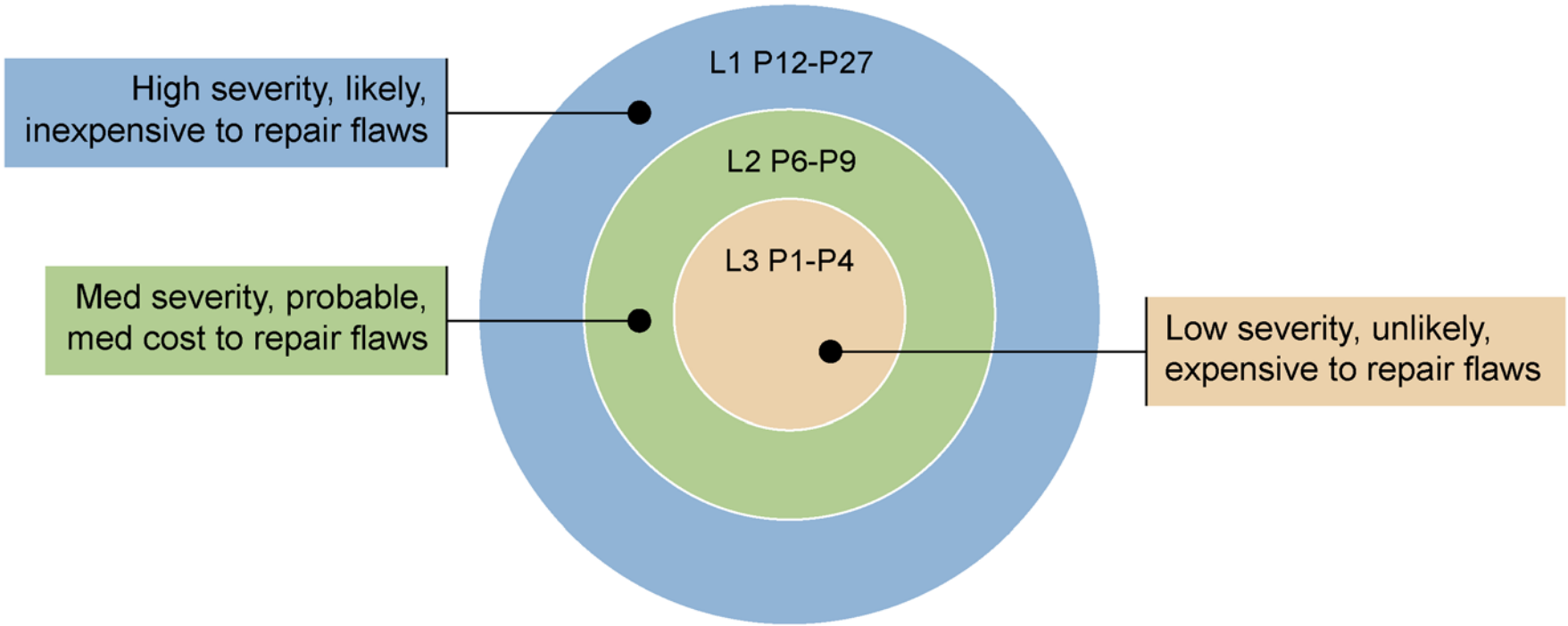
Likelihood – how likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability?

Value	Meaning
1	unlikely
2	probable
3	likely

Cost – the cost of mitigating the vulnerability.

Value	Meaning	Detection	Correction
1	high	manual	manual
2	medium	automatic	manual
3	low	automatic	automatic

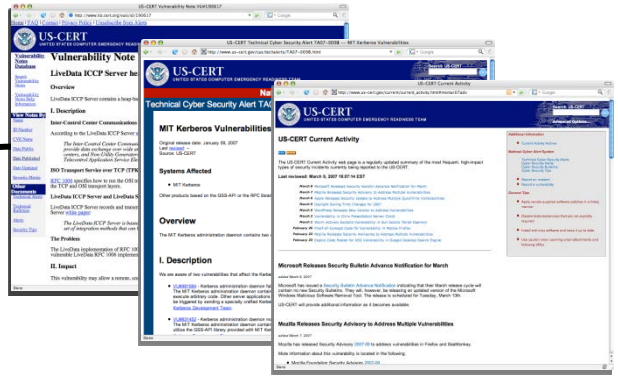
Priorities and Levels



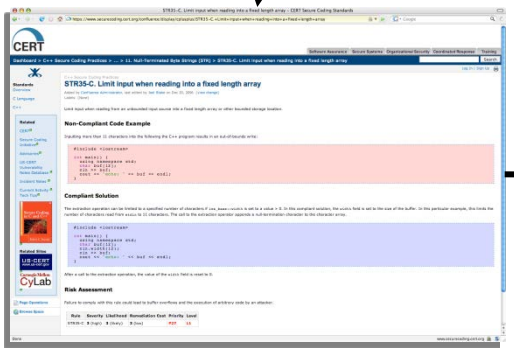
CERT Mitigation Information

Vulnerability Note VU#649732

This vulnerability occurred as a result of failing to comply with rule FIO30-C of the CERT C Programming Language Secure Coding Standard.



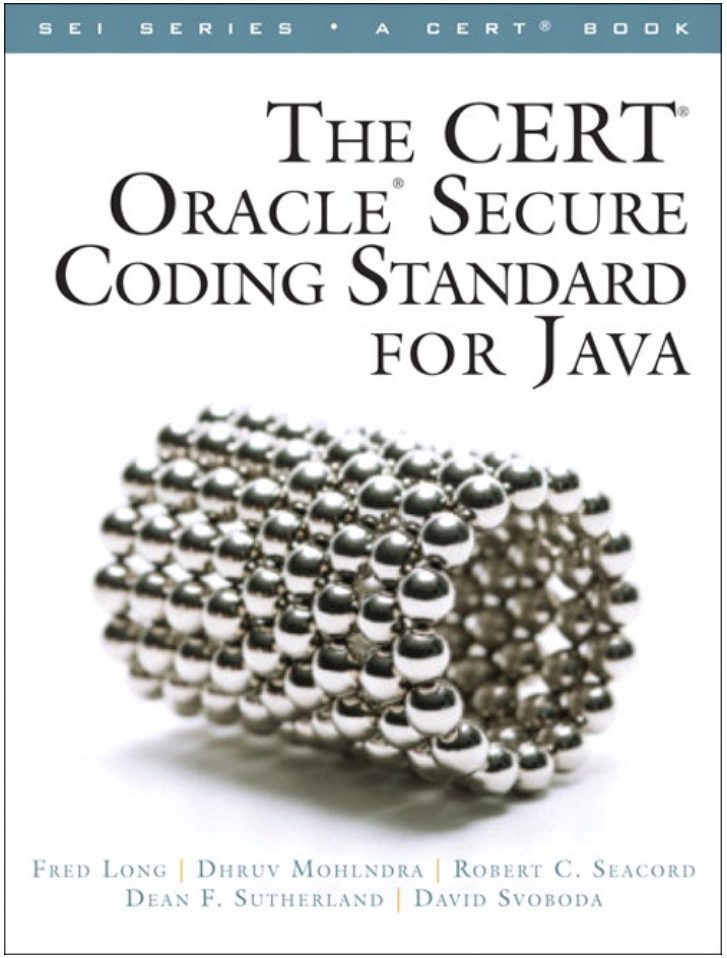
US CERT Technical Alerts



CERT Secure Coding Standard

Examples of vulnerabilities resulting from the violation of this recommendation can be found on the CERT website .

Secure Coding Standard for Java



"In the Java world, security is not viewed as an add-on a feature. It is a pervasive way of thinking. Those who forget to think in a secure mindset end up in trouble. But just because the facilities are there doesn't mean that security is assured automatically. A set of standard practices has evolved over the years. ***The Secure® Coding® Standard for Java™*** is a compendium of these practices. These are not theoretical research papers or product marketing blurbs. This is all serious, mission-critical, battle-tested, enterprise-scale stuff."

–**James A. Gosling,** Father of the Java Programming Language

Issues Not Addressed

Design and Architecture

- This standard assumes that the design and architecture of the product is secure, that is, that the product is free of design-level vulnerabilities that would otherwise compromise its security.

Tools.

- As a federally funded research and development center (FFRDC), the Software Engineering Institute (SEI) is not in a position to recommend particular vendors or tools to enforce the restrictions adopted.
- Users of The CERT Oracle Secure Coding Standard for Java are free to choose tools; vendors are encouraged to provide tools to enforce these rules.

Issues Not Addressed

Content

- This coding standard does not address concerns specific to only one Java based platform but applies broadly to all platforms.
- For example, rules that are applicable to Java Micro Edition (ME) or Java Enterprise Edition (EE) alone and not to Java SE are typically not included.
- Within Java SE, APIs that deal with the user interface (User Interface Toolkits) or with the web interface for providing features such as sound, graphical rendering, user account access control, session management, authentication, and authorization are beyond the scope of this standard.
- However, this does not preclude the standard from discussing networked Java systems given the risks associated with improper input validation and injection flaws and suggesting appropriate mitigation strategies.

Issues Not Addressed

Coding Style

- Coding style issues are subjective; it has proven impossible to develop a consensus on appropriate style rules.
- Consequently, The CERT Oracle Secure Coding Standard for Java recommends only that the user define style rules and apply those rules consistently; requirements that mandate use of any particular coding style are deliberately omitted.
- The easiest way to consistently apply a coding style is with the use of a code formatting tool. Many integrated development environments (IDEs) provide such capabilities.

Scope

The CERT Oracle Secure Coding Standard for Java focuses on the Java Standard Edition 6 Platform (Java SE 6) environment and includes rules for secure coding using the Java programming language and libraries.

The Java Language Specification (3rd edition) [JLS 2005] prescribes the behavior of the Java programming language and served as the primary reference for the development of this standard.

This coding standard also addresses new features of the Java SE 7 Platform, primarily, as alternative compliant solutions to secure coding problems that exist in both the Java SE 6 and Java SE 7 platforms.

Source Code Analysis Laboratory

The CERT Source Code Analysis Laboratory (SCALe) is an operational capability for application conformance testing against one of CERT's secure coding standards.

- A detailed report of findings is provided to the customer to repair
- After the developer has addressed these findings, the product version is certified as conforming to the standard
- The certification is published in a registry of certified systems

SCALe Process Overview

1. Client contacts CERT. The process is initiated when a client contacts CERT with a request to certify a software system.

2. CERT communicates requirements. CERT communicates requirements to the customer, including (1) selection of secure coding standard(s) to be used, (2) a buildable version of the software to be evaluated, and (3) a build engineer.

3. Client provides buildable software. Client selects standard(s), provides a buildable version of the software to be evaluated, and identifies the build engineer, who is available to respond to build questions for the system.

4. CERT selects tool set. CERT chooses and documents the tool set to be used and procedures for using that tool set in evaluation of the system.

5. CERT analyzes source code and generates conformance test report. CERT evaluates the system against specified standard(s) and provides the conformance test results to the client and, if the system is found to be conforming, issues a certificate and terminates the conformance testing process.

6. Client repairs software. Client has the opportunity to repair nonconforming code. Client sends system back to CERT for final evaluation.

7. CERT issues conformance tests results and certificate. CERT reevaluates the system using the tools and procedures used in the initial assessment. CERT provides the conformance test results to the client and, if the system is found to be conforming, a certificate.

Government Demand

CERT secure coding initiative has performed source code assessments for various government agencies.

The Application Security and Development Security Technical Implementation Guide (STIG)

- is being specified in DoD acquisition programs' Request for Proposals (RFPs).
- provides security guidance for use throughout an application's development lifecycle.

Section 2.1.5, “Coding Standards” of the Application Security and Development STIG identifies the following requirement:

(APP2060.1: CAT II) The Program Manager will ensure the development team follows a set of coding standards."

Industry Demand

Conformance with CERT Secure Coding Standards can represent a significant investment by a software developer, particularly when it is necessary to refactor or modernize existing software systems.

However, it is not always possible for a software developer to benefit from this investment, because it is not always easy to market code quality.

A goal of conformance testing is to provide an incentive for industry to invest in developing conforming systems.

- perform conformance testing against CERT secure coding standards
- verify that a software system conforms with a CERT secure coding standard
- use CERT “seal” when marketing products
- maintain a certificate registry with the certificates of conforming systems

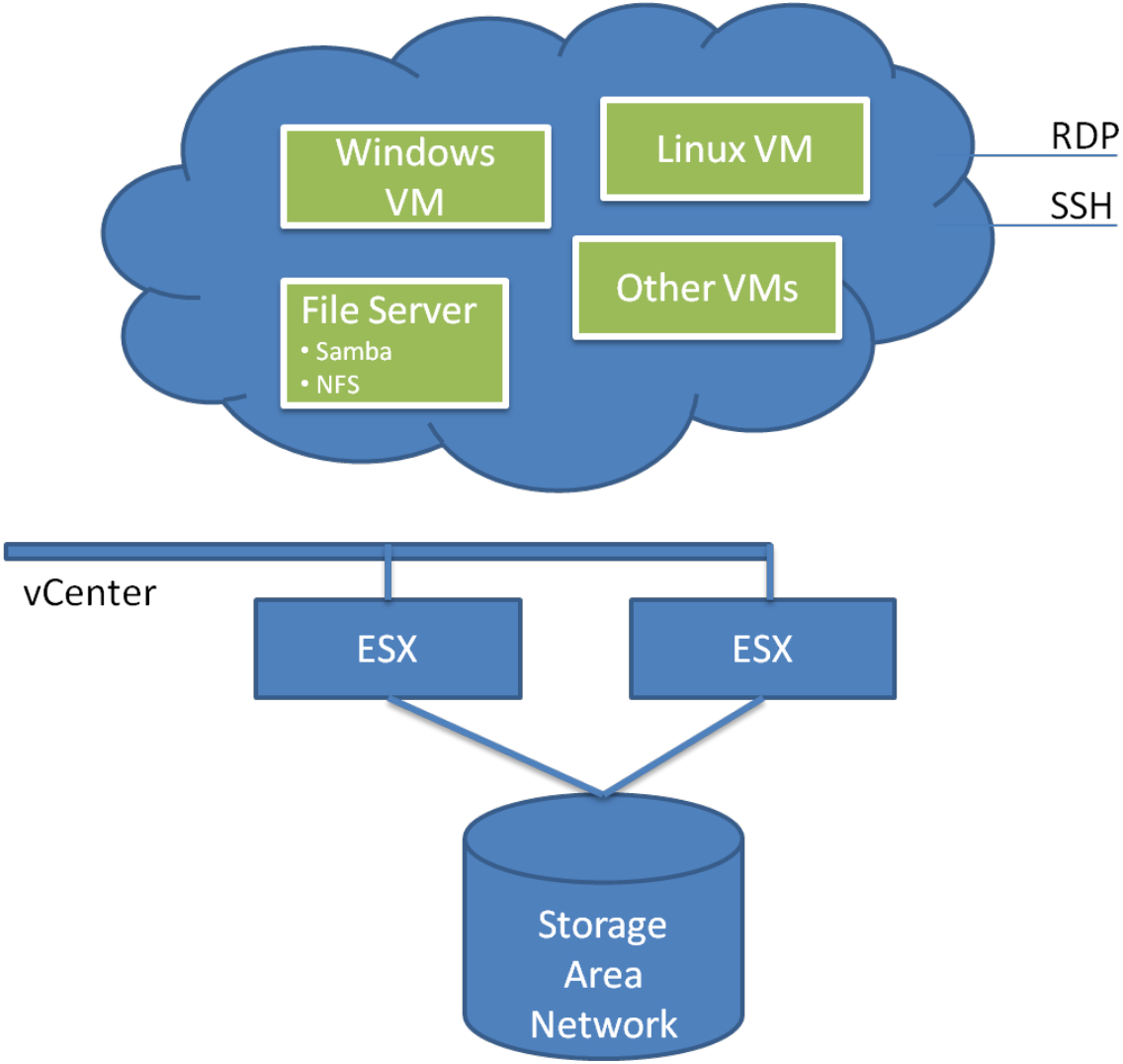
CERT SCALe Seal

Developers of software that has been determined by CERT to conform to a secure coding standard may use the to describe the conforming software on the developer's website.

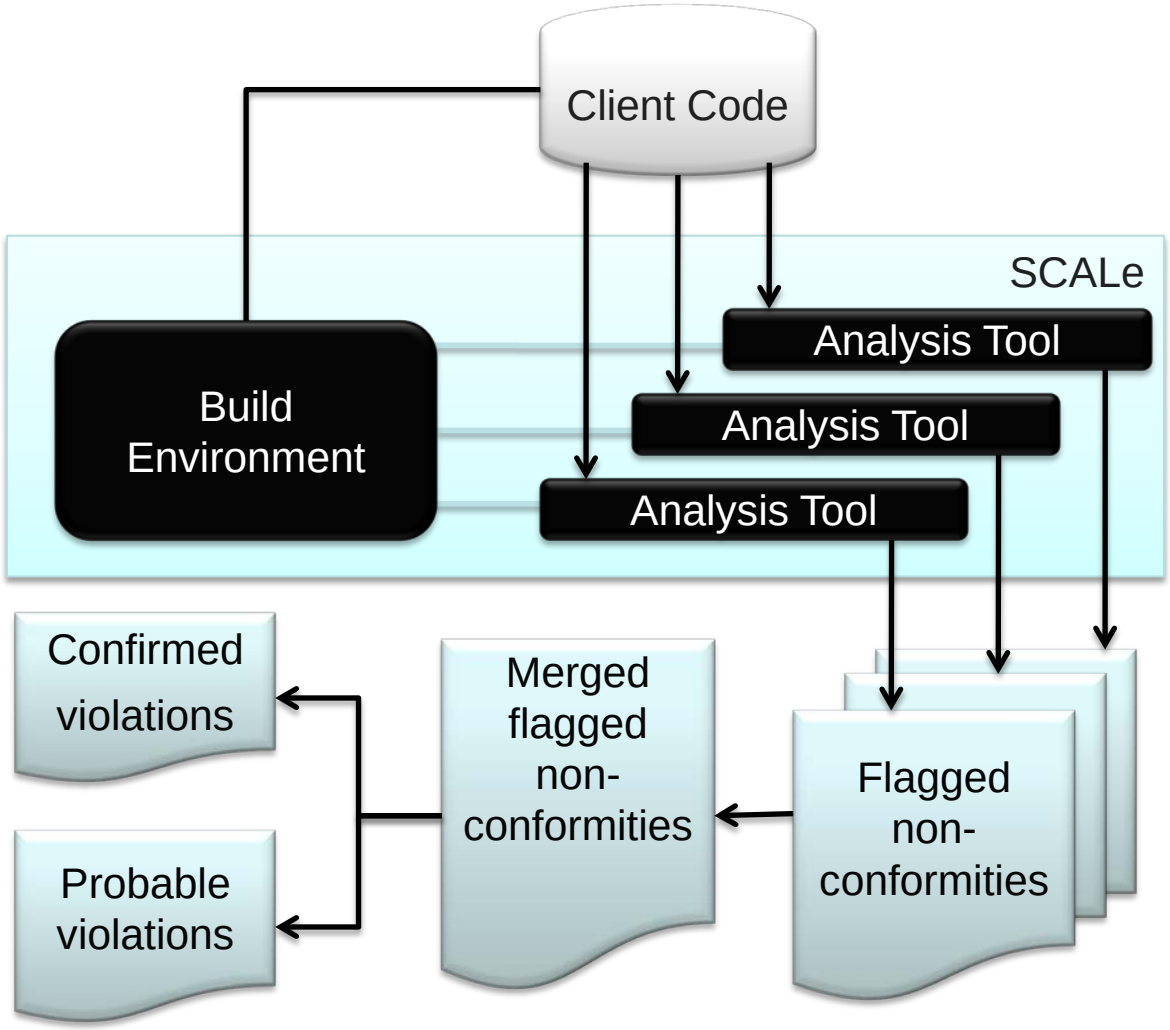
The seal must be specifically tied to the software passing conformance testing and not applied to untested products, the company, or the organization.

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.

Source Code Analysis Laboratory



Conformance Testing Process



Conformance Testing

The use of secure coding standards defines a proscriptive set of rules and recommendations to which the source code can be evaluated for compliance.

For each secure coding standard, the source code is certified as provably nonconforming, conforming, or provably conforming against each guideline in the standard:

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	Finally, the code is provably conforming if the code has been verified to adhere to the rule in all possible cases.

Evaluation violations of a particular rule ends when a “provably nonconforming” violation is discovered.

False Positives

Static analysis tools frequently generate large numbers of false positives

A statistical sampling approach (lot tolerance percent defective (LTPD)) is used to select a random sample of flagged nonconformities from a given bucket for further investigation.

The following table shows the number of samples evaluated based on the number of flagged nonconformities per rule and LQ (by default, SCALe uses 2.0%).

Flagged Nonconformities per Rule	Nominal Limiting Quality in Percent (LQ)					
	0.5%	0.8%	1.25%	2.0%	3.15%	5.0%
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
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 1200	380	255	170	125	125	80

[1] If the required sample size is greater than the bucket size, then the sample size is the bucket size.
[2] At this LQ value and bucket size, the sampling plan actually would allow one observed true positive in the sample investigated, but the SCALe analyst will remain using the zero observed true positive rule to decide if the bucket is acceptable or not.
[3] Same comment as the previous footnote

Analysis Procedure

1. Identify the Nominal Quality Level (LQ) desired for the security analysis.
2. Group flagged nonconformities for a given security rule into buckets.
3. Use the table to identify the required sample size (**n**). Note that at the 2% LQ, all flagged nonconformities are investigated if the bucket totals 50 or less.
4. Select the specified number (**n**) of random nonconformities from the flagged nonconformities in the bucket.
5. Investigate each flagged nonconformity in the sample to determine whether it is a false or true positive flagged nonconformity and label accordingly.
6. If all flagged nonconformities in the sample are false positives, all 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. The remaining flagged nonconformities in the bucket that were not investigated are categorized as *probable violations*.

Deviation Procedure

Strict adherence to all rules is unlikely; 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.

For More Information

Visit CERT® web sites:

http://www.cert.org/secure-coding/

https://www.securecoding.cert.org/

Contact

Robert C. Seacord

rcs@cert.org

(412) 268-7608

Contact CERT:

Software Engineering Institute

Carnegie Mellon University

4500 Fifth Avenue

Pittsburgh PA 15213-3890

USA