SEI WEBINAR SERIES | Keeping you informed of the latest solutions

Software Engineering Institute | Carnegie Mellon University

Carnegie Mellon University

This video and all related information and materials ("materials") are owned by Carnegie Mellon University. These materials are provided on an "as-is" "as available" basis without any warranties and solely for your personal viewing and use.

You agree that Carnegie Mellon is not liable with respect to any materials received by you as a result of viewing the video, or using referenced websites, and/or for any consequences or the use by you of such materials.

By viewing, downloading, and/or using this video and related materials, you agree that you have read and agree to our terms of use (www.sei.cmu.edu/legal/).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Distribution Statements

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0003519

Secure Coding Best Practices

Robert Schiela

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

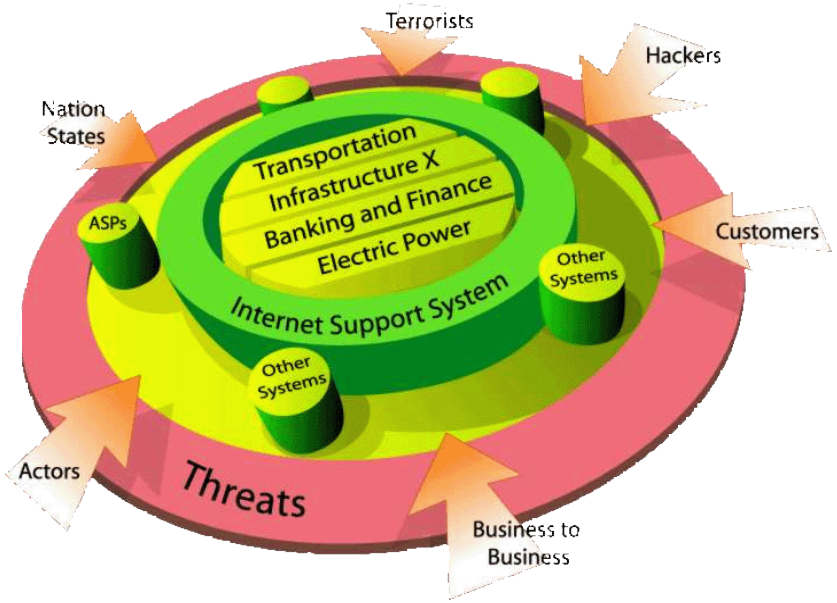
Why Software Security?

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, and continuity
- critical infrastructures & government



Most Vulnerabilities Are Caused by Programming Errors

64% of the vulnerabilities in the NIST National Vulnerability Database due to programming errors

- 51% of those were due to classic errors like buffer overflows, cross-site scripting, injection flaws

Top vulnerabilities include

- Integer overflow
- Buffer overflow
- Uncontrolled Format String
- Missing authentication
- Missing or incorrect authorization
- Reliance on untrusted inputs (aka tainted inputs)

Sources: Heffley/Meunier: Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?; cwe.mitre.org/top25 Jan 6, 2015

Secure Software Development

Secure software development starts with understanding insecure coding practices, and how these may be exploited.

Insecure designs can lead to “intentional errors”, that is, the code is correctly implemented but the resulting software contains a vulnerability.

Secure designs require an understanding of functional and non-functional software requirements.

Secure coding requires an understanding of implementation specifics.

Sources of Software Insecurity

Absent or minimal consideration of security during all life cycle phases

Complexity, inadequacy, and change

Incorrect or changing assumptions

Not thinking like an attacker

Flawed specifications & designs

Poor implementation of software interfaces

Unintended, unexpected interactions

- with other components
- with the software's execution environment

Inadequate knowledge of secure coding practices

Unspecified and Undefined Behaviors

implementation-defined behavior - Unspecified behavior whereby each implementation documents how the choice is made.

unspecified behavior - Behavior for which the standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance.

undefined behavior - Behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which the standard imposes no requirements. An example of undefined behavior is the behavior on integer overflow.

From ISO/IEC 9899-1999.

Polling Question

Does your organization use a coding standard for security?

- Yes
- No
- Maybe?

Polling Question

What programming language do you work with most in your organization?

- Assembly
- C
- C++
- C#
- Java
- Java-Script
- Objective-C
- Perl
- PHP
- Python
- PL/SQL or SQL
- Ruby
- Swift
- Visual Basic
- Other
- Little to none developed in-house

Adopting Secure Coding Practices

Secure Coding Infrastructure

- Defining Secure Coding Practices
- Influencing Language Standards
- Influencing Tool Vendors

People

- Training

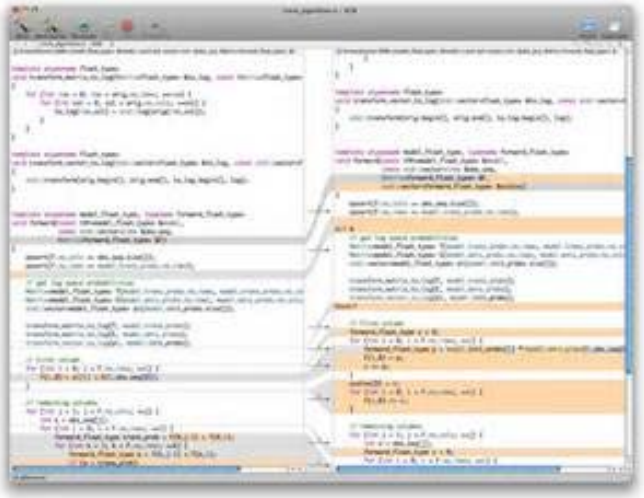
Processes

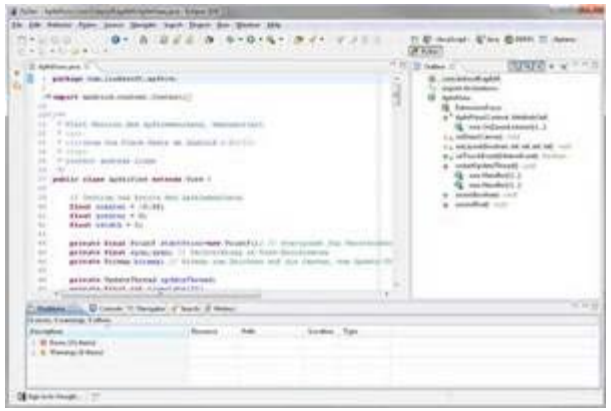
- Coding Standards and Security Standards, Testing

Technology

- Tools: IDE's and Analyzers
- Automated transformation and remediation

Tools encourage application of secure coding





Moving rules into IDEs improves application of secure coding:

- Early feedback corrects errors on introduction.
- Exceptions are understood in context.

Adoption of secure coding IDEs

- help deploy tools
- training on tools
- extend tools to meet targeted needs

Conformance Testing

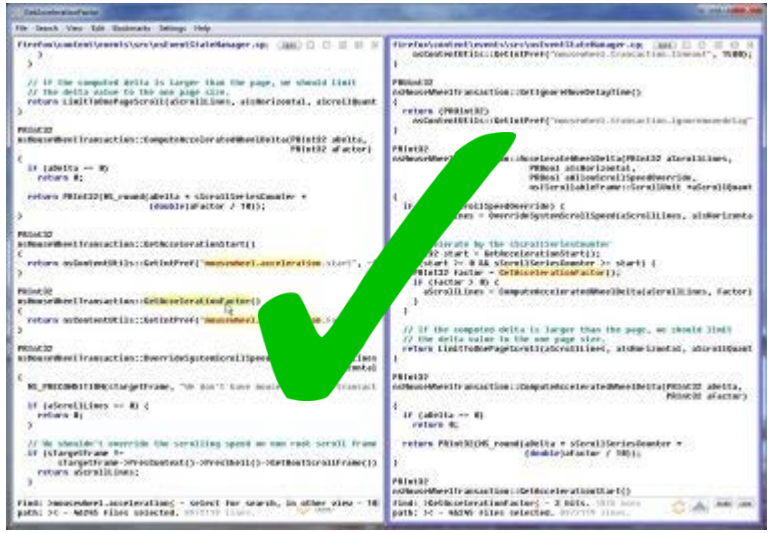
The use of secure coding standards defines a proscriptive set of rules and recommendations to which the source code can be evaluated for compliance.

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.

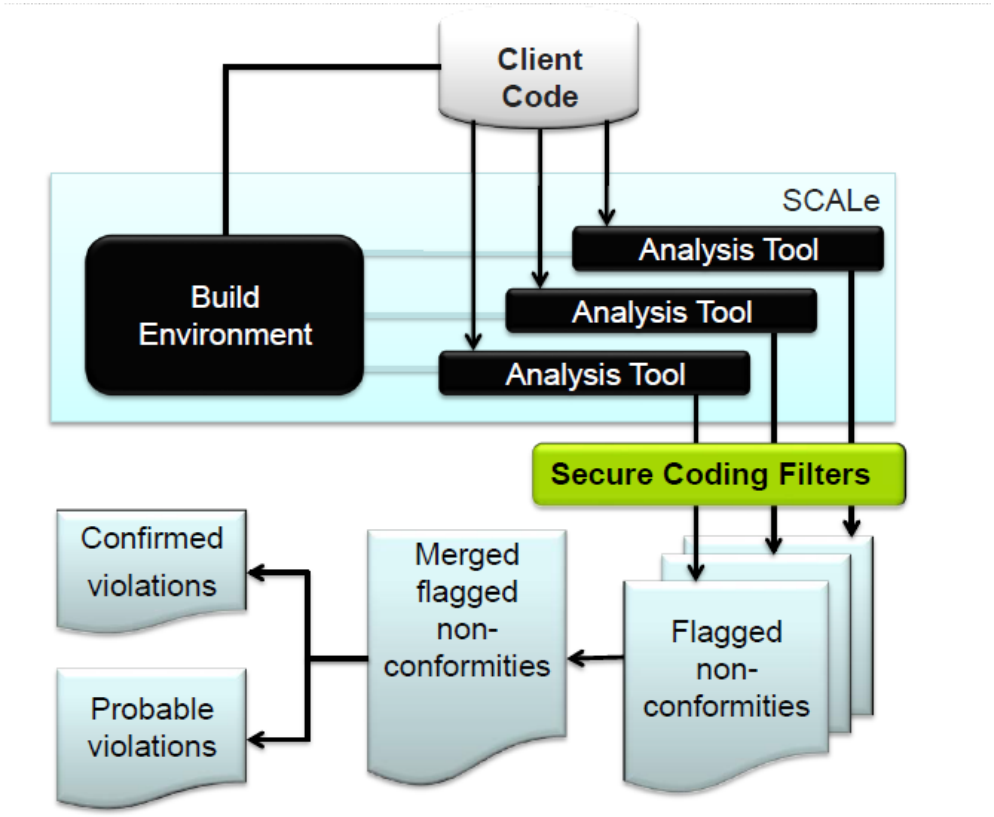
Static Testing – Source code analysis tools



Secure Code Analysis Laboratory (SCALe)

- C, C++, Java, PERL, Python, Android rule conformance checking
- Thread safety analysis
- Information flows across Android applications
- Operating system call flows

SCALe Multitool evaluation



Improve expert review productivity by focusing on high priority violations

Filter select secure coding rule violations

- Eliminate irrelevant diagnostics
- Convert to common CERT Secure Coding rule labeling

Single view into code and all diagnostics

Maintain record of decisions

Polling Question

What testing does your organization perform on your software?

- Static Analysis
- Dynamic Analysis
- Both
- None

Supply Chain Software

Install Security Updates.

Test Source Code.

Review vendors' security and software assurance practices and results.

Request reports from their own testing or request independent security reviews and testing.

Test Binaries.

Apply Defense in Depth – only enable features that are required, and protect them.

Isolate critical and non-critical services and data.

Perform penetration testing.

Install an independent monitoring system.

Contact Information

Robert Schiela
Technical Manager, Secure Coding
Telephone: +1 412.268.3637
Email: rschiela@cert.org

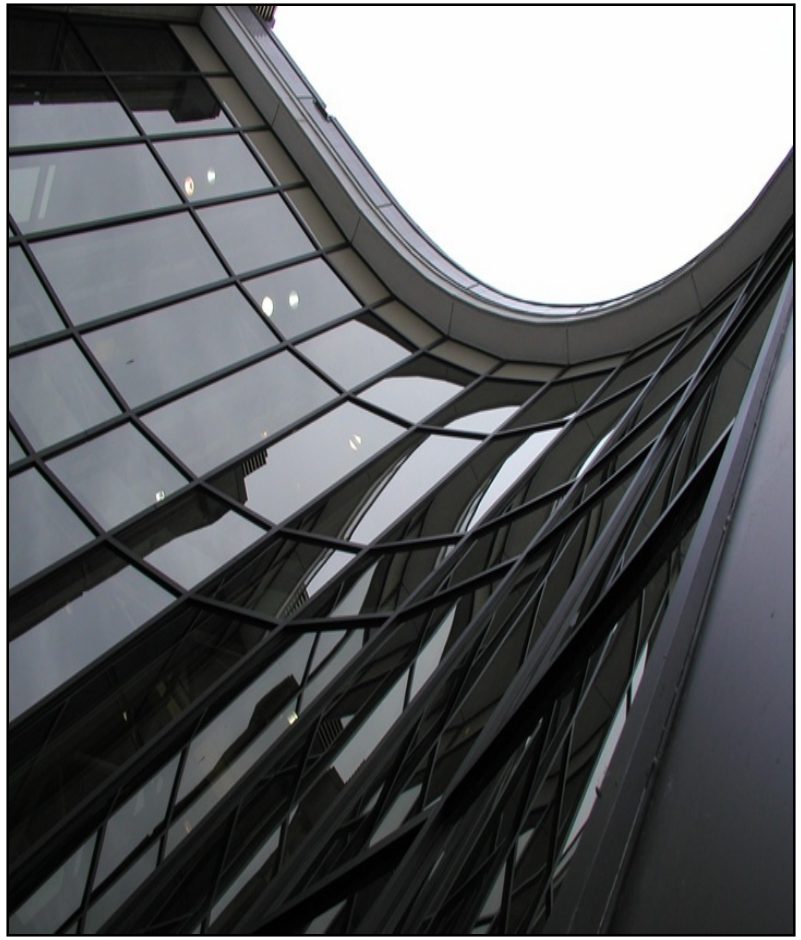
Web Resources

http://www.sei.cmu.edu/

http://www.cert.org/

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

http://securecoding.cert.org/
(SEI CERT Secure Coding Standards Wiki)



Software Engineering Institute | Carnegie Mellon