TSP Secure

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Noopur Davis, Philip L. Miller, William R. Nichols, and Robert C. Seacord

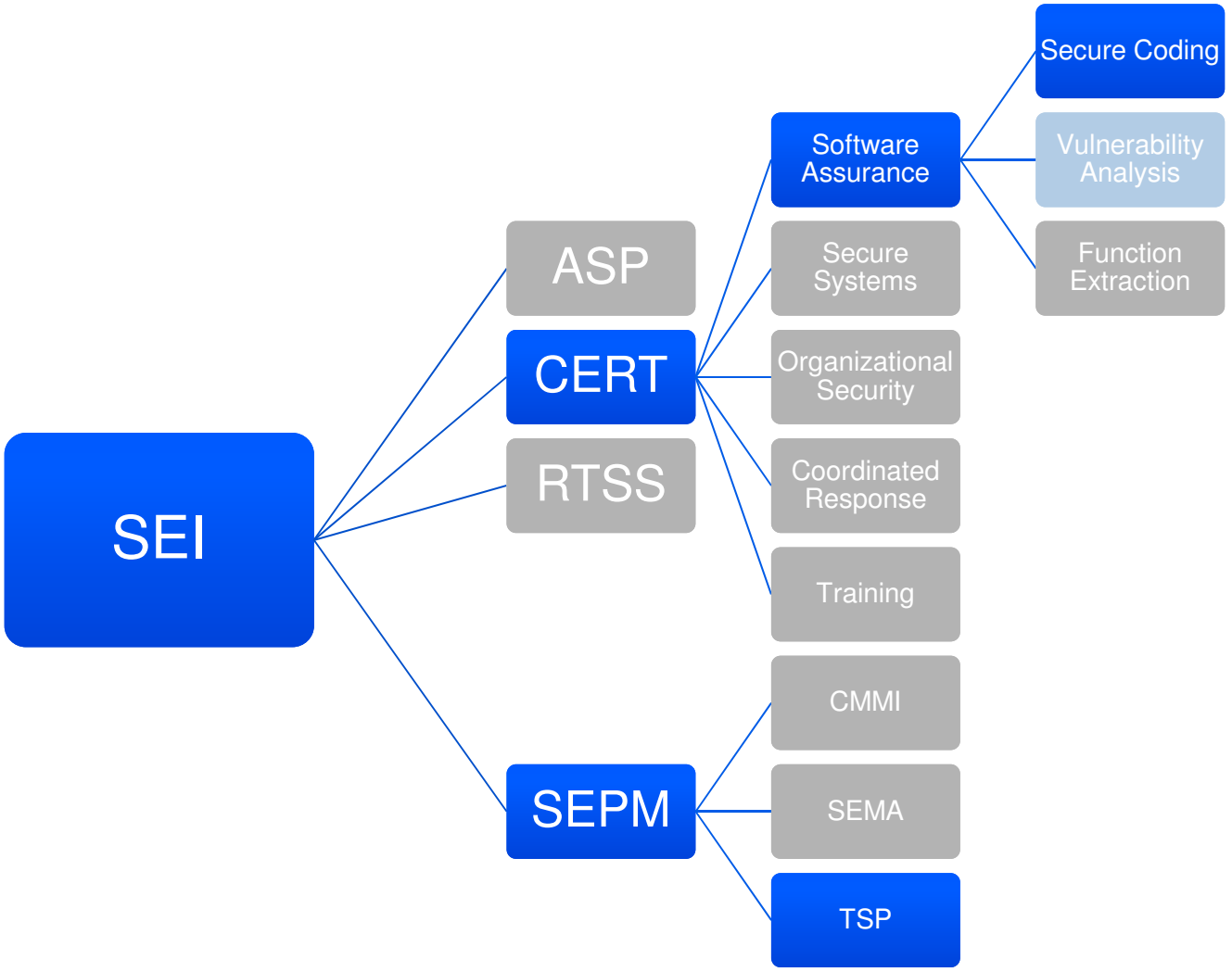
23 September 2009

Software Engineering Institute | Carnegie Mellon

Outline

- Initiative in a nutshell
- Assumptions about the audience
- Vulnerability and exploits
- Advances in secure coding
- Why TSP
- Prior work
- Integration
- Next steps

Initiative in a nutshell-1



Initiative in a nutshell-2

Objective

Build software that is free from known vulnerabilities; vulnerabilities that – if exploited – enable a determined adversary to violate security policies. Examples include

- running arbitrary code
- accessing sensitive information
- denying services to legitimate users

Method

Build software correctly in the first place.

… for all the usual reasons only more so.

Assumptions about the audience

- Have good general TSP knowledge
- Are aware of software exploitation
  - May have few details
  - May be out of date
- Know little about prior TSP-Secure work

Secure Programming is a Challenge

The C Standard defines **undefined behavior** as:

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.

Undefined Behaviors in C

Undefined behaviors are identified in the standard:

- If a "shall" or "shall not" requirement is violated, and that requirement appears outside of a constraint, the behavior is undefined.
- Undefined behavior is otherwise indicated in this International Standard by the words "undefined behavior"
- by the omission of any explicit definition of behavior.

There is no difference in emphasis among these three; they all describe "behavior that is undefined".

C99 Annex J.2, "Undefined behavior," contains a list of explicit undefined behaviors in C99.

Undefined Behaviors in C

Behaviors are classified as "undefined" by the standards committees to:

- give the implementer license not to catch certain program errors that are difficult to diagnose;
- avoid defining obscure corner cases which would favor one implementation strategy over another;
- identify areas of possible conforming language extension: the implementer may augment the language by providing a definition of the officially undefined behavior.

Implementations may

- ignore undefined behavior completely with unpredictable results
- behave in a documented manner characteristic of the environment (with or without issuing a diagnostic)
- terminate a translation or execution (with issuing a diagnostic).

Fun With Integers

```
char x, y;
x = -128;
y = -x;

if (x == y) puts("1");
if ((x - y) == 0) puts("2");
if ((x + y) == 2 * x) puts("3");
if (((char)(-x) + x) != 0) puts("4");
if (x != -y) puts("5");
```

Lesson: Process must be supplemented with a strong fundamental knowledge of the language and environment

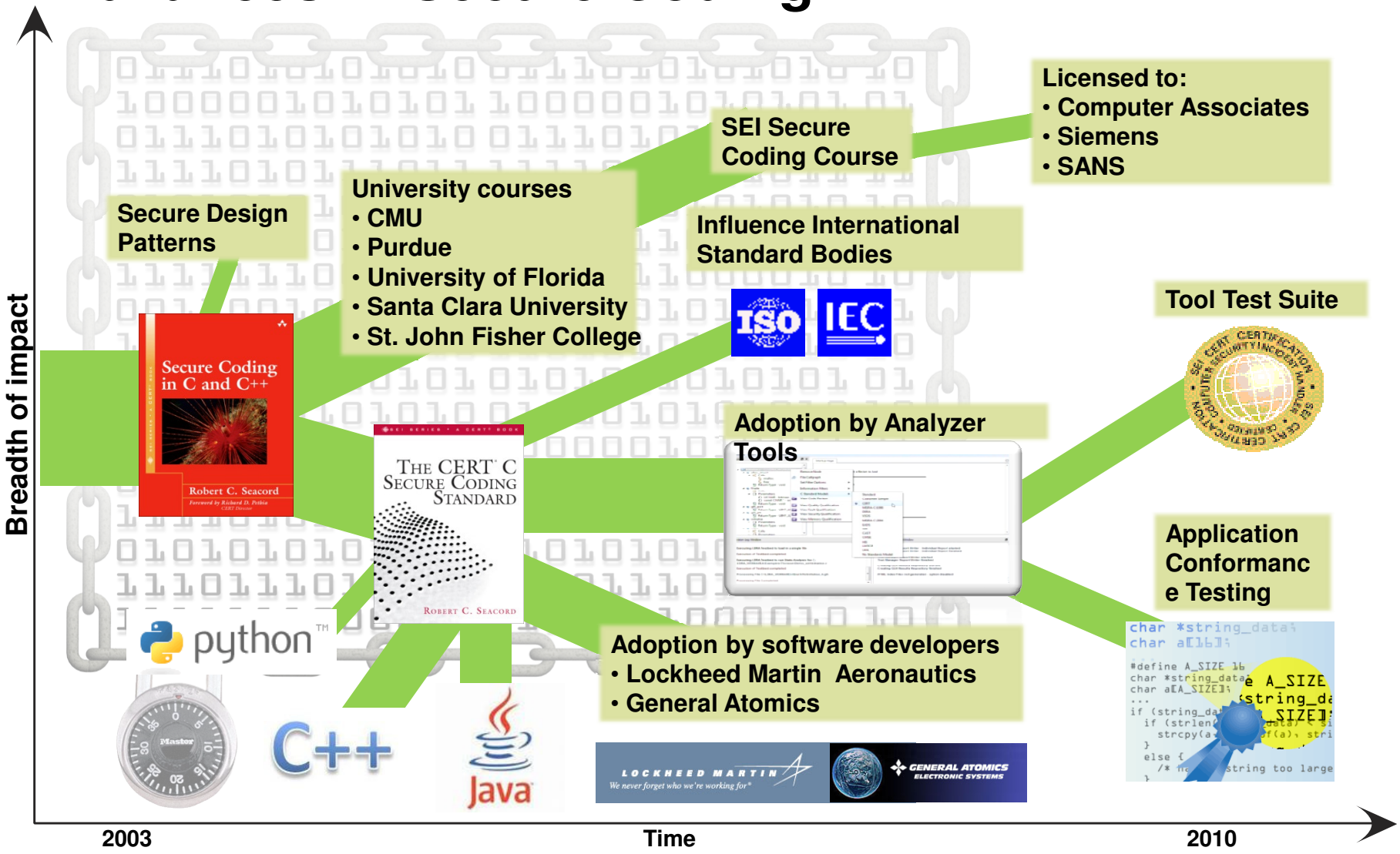
Integer operations that may result in overflow and undefined behavior

Op	Overflow	Op	Overflow	Op	Overflow
+	✓	*=	✓	&	
-	✓	/=	✓	|	
*	✓	%=	✓	^	
/	✓	<<=	✓	~	
%	✓	>>=	✓	!	
++	✓	&=		un +	
--	✓	|=		un -	✓
=		^=		<	
+=	✓	<<	✓	>	
-=	✓	>>	✓	etc.	

Integer Overflow Vulnerabilities

In 2007, MITRE reported that integer overflow, barely in the top 10 overall in the years preceding the report, was the number two issue as reported in operating system (OS) vendor advisories (after buffer overflow, which may also be caused by integral security issues).

Advances in Secure Coding



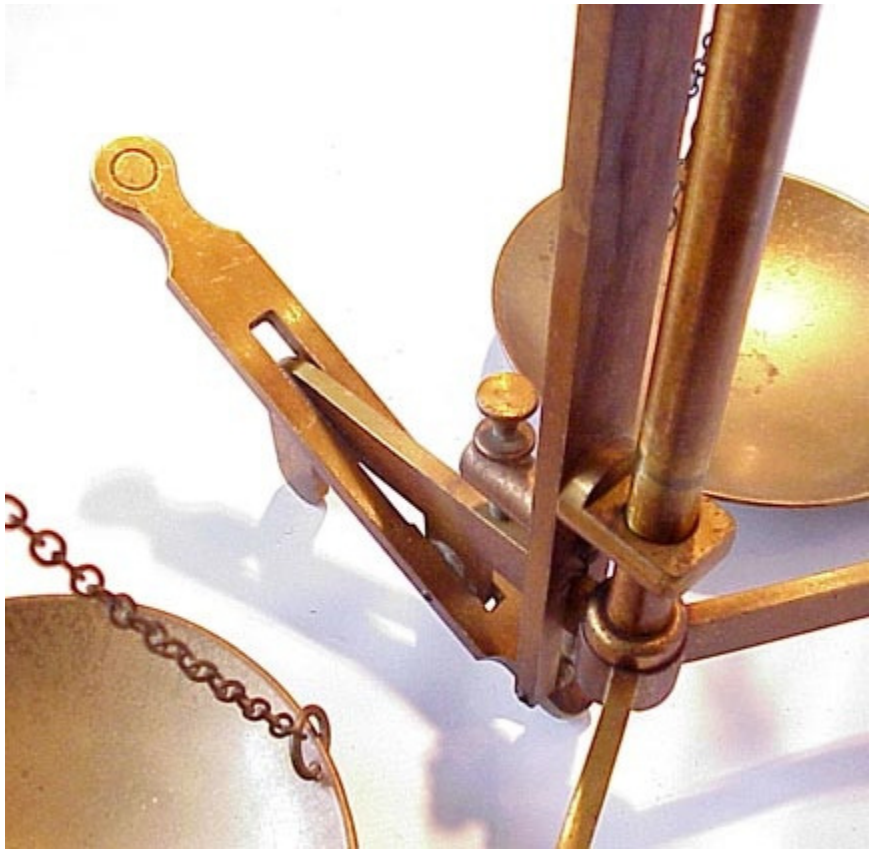
CERT SCALe (Source Code Analysis Lab)

Satisfy demand for source code assessments for both government and industry organizations

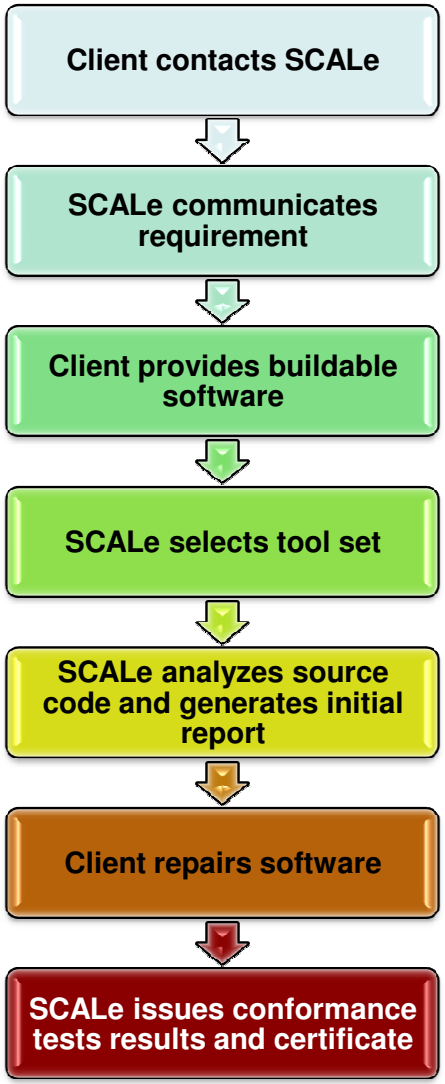
Assess source code against one or more secure coding standards.

Provided a detailed report of findings

Assist customers in developing certifiably conforming systems



Conformance Testing



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

INT30-C.	Provably nonconforming
INT31-C.	Documented deviation
INT32-C.	Conforming
INT33-C.	Provably Conforming

Why TSP?

- Produces nearly defect-free software
- Has frameworks for planning, measurement, and quality management
- Supports the use of processes and standards
  - Planning for quality,
  - Tracking and managing the development plan,
- Empowers self-directed teams committed to common goals, and management and mitigation of risks
- Builds developer training into the plan
- Reinforces training with reviews, inspections, and tools
- Has predictive capability

Prior work

	Post code complete defects	
Phase	Prior similar release	TSP-Secure release
Integration Test	237	4
System Test	473	3
User Acceptance Test	153	10
Security code defects	Data not available	0
Total Defects	1072	17

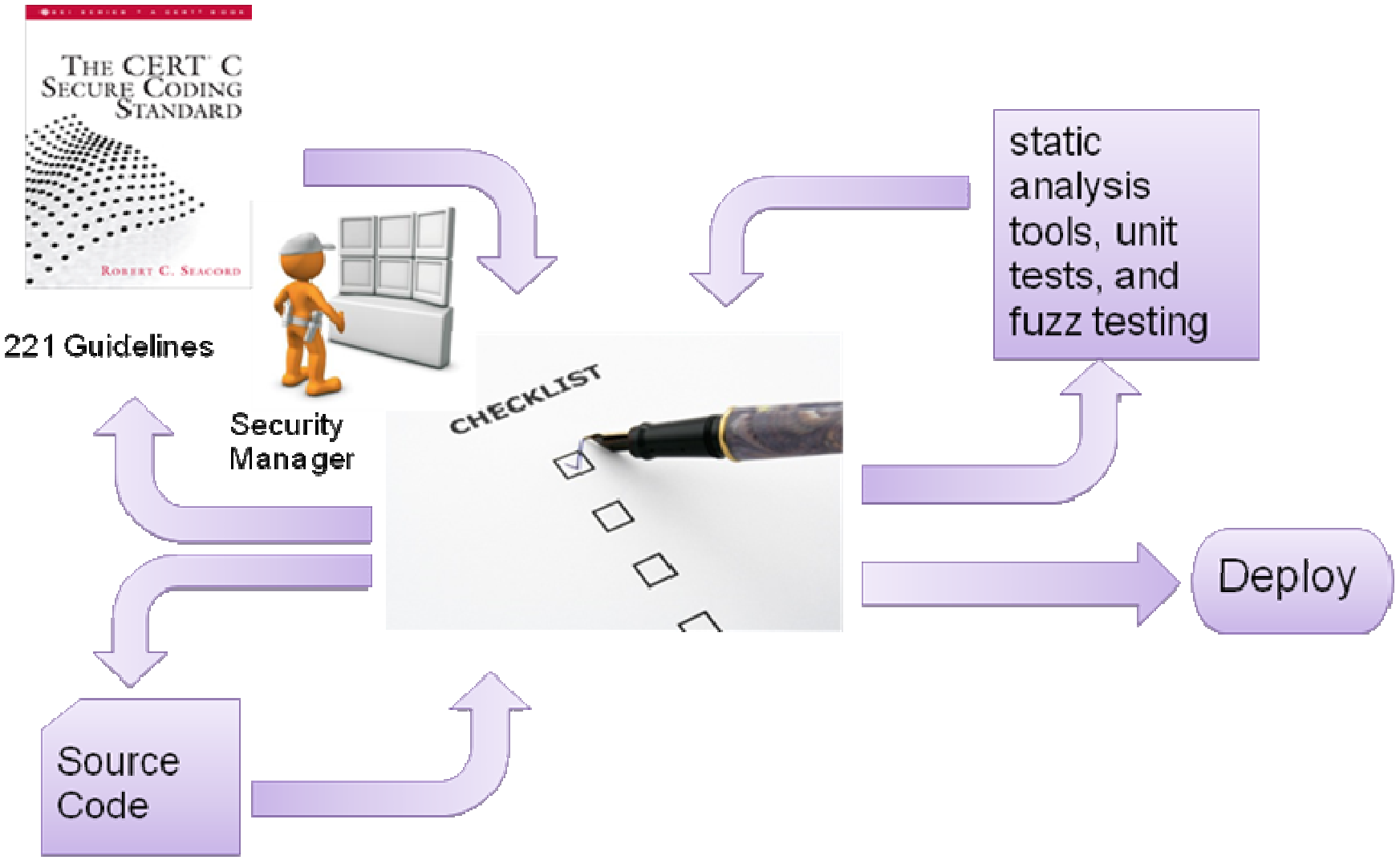
- 2002 Microsoft Secure Code Project
- 8-person software development team
- Created 30 thousand lines of new and modified code in 7 months
- Resulted in the initial TSP-Secure

Deliverables from Prior Work

. 2-day TSP-Secure workshop

- Overview of Common Causes of vulnerabilities
- In-Depth look at top causes of vulnerabilities
  - Buffer Overflow
  - SQL Injection
  - Cross-site Scripting
  - Race Conditions
  - Etc..
- Design patterns for vuls
  - State machine verification
  - Updated DESIGN script
- Secure code inspections
  - New REVIEW script

Integration-1



Integration-2

Integration-3

- Selection of a secure coding standard during requirements
- Train all developers on appropriate secure coding
- Train engineers in source code analysis.
- techniques prior to project launch
- Define a new team role, Security Manager
- Specify and script additional pre-launch meetings
- Modify and script existing launch meetings
- Integrate the use of static analysis tools

Next Steps

- Continue the fundamental technical work on integrating secure coding and TSP
- Pilot the existing package with selected organizations on a few projects
- Publish results
- Revise TSP-Secure based on outcomes of pilot projects.

Contact Information

Philip Miller
Sr. Member of Operational Staff
Program Development and Transition
Telephone: +1 412-268-3560
Email: pmiller@sei.cmu.edu

World Wide Web: www.sei.cmu.edu

U.S. mail:
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA