

Construction and Implementation of CERT Secure Coding Rules Improving Automation of Secure Coding

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Mark Sherman, PhD
Technical Director, CERT
mssherman@sei.cmu.edu

Aaron Ballman
Software Security Engineer, CERT
aballman@cert.org





Copyright 2016 Carnegie Mellon University

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.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

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.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material 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 use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered marks of Carnegie Mellon University.

DM-0003814



Agenda

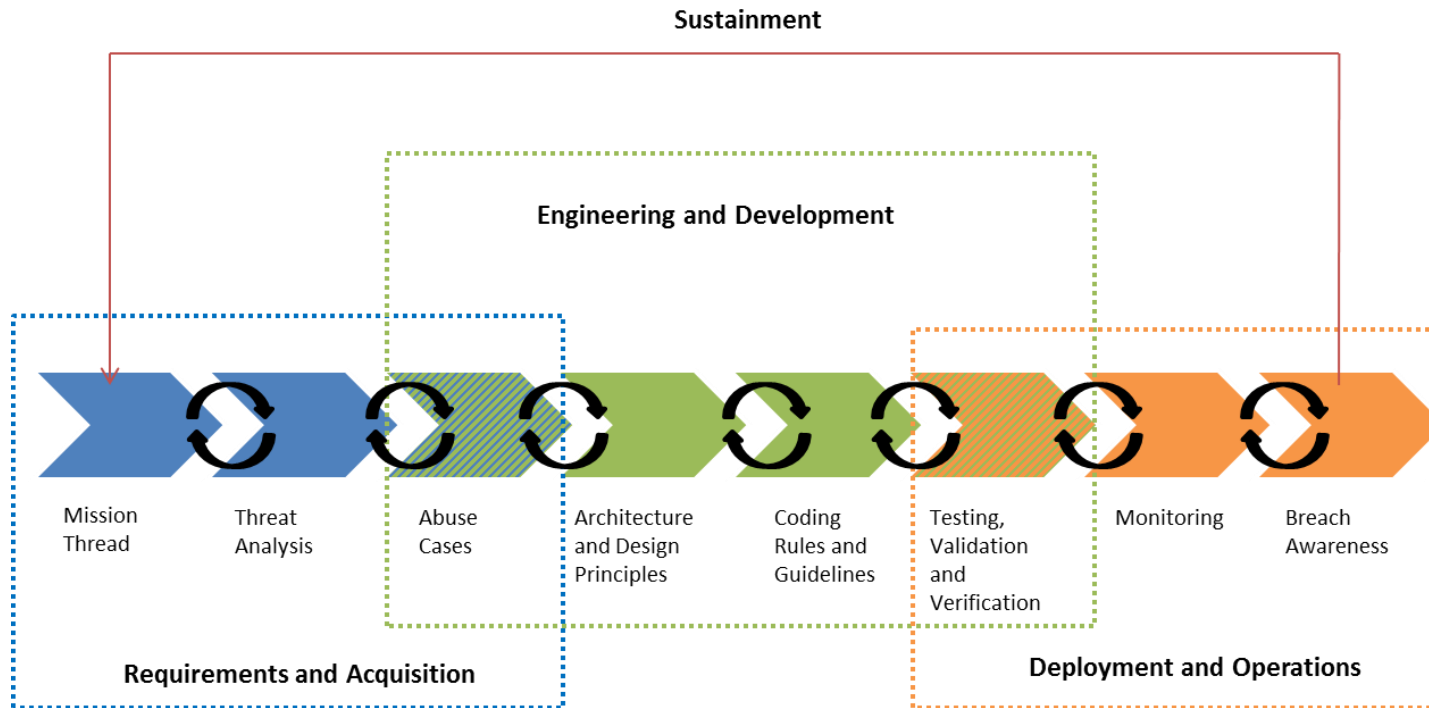


- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**



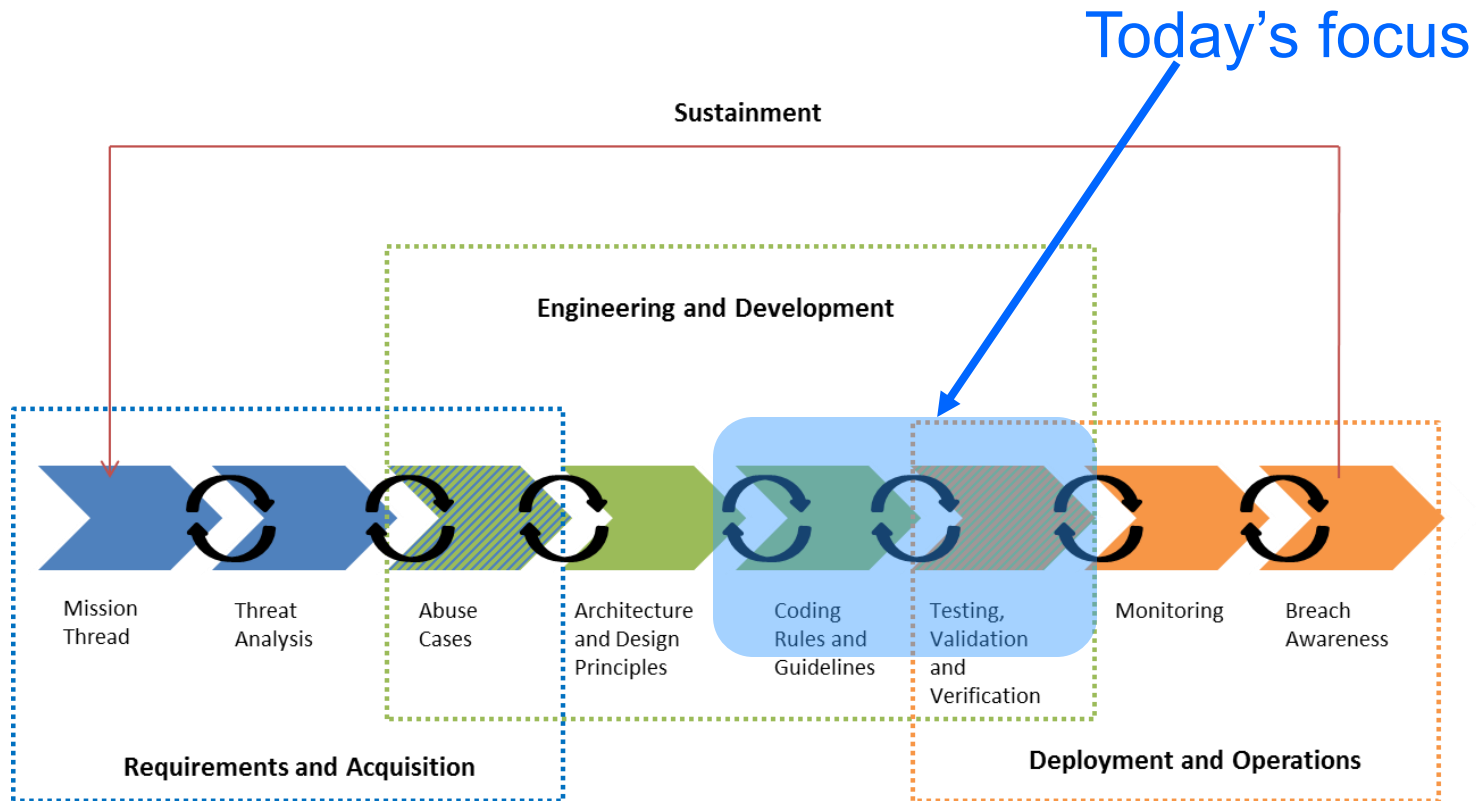


Security is a lifecycle issue

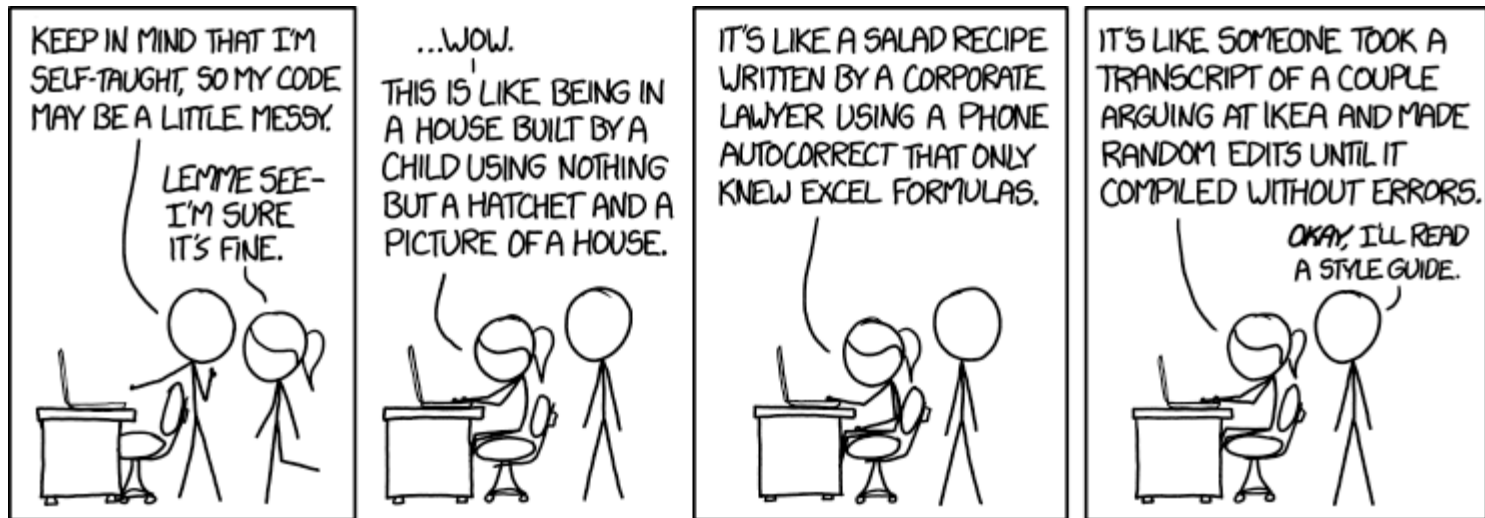




Security is a lifecycle issue



Code security quality reviews generally reveal problems



Source: <http://xkcd.com/1513/>



Code security quality reviews generally reveal problems – that manifest as vulnerabilities



Source: <http://xkcd.com/1695/>





Most Vulnerabilities Are Caused by Programming Errors

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

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

Top 25 CWE includes

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

Sources: Heffley/Meunier (2004): Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?; cwe.mitre.org/top25 Jan 6, 2015





Agenda



- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**



Another crises occurs

BREACH

Report: Growing risk of cyber attacks on banks
MIAMI HERALD Business Breaking News
Posted on Tuesday, 05.06.14
The report, issued Monday by Gov. Andrew Cuomo and the state's Department of Financial Services, found that a majority of the 154 attacks occurred in the past three years.
The attacks involved the use of accounts, seize data and steal information.
Cuomo says he's directing banks to take steps to make themselves more vulnerable they are to attack.

Defense Companies Facing Array of New Cyberthreats
NATIONAL DEFENSE
Cybersecurity
This Month
January
February
March
April
May
June
July
Defense Companies Facing Array of New Cyberthreats (UPDATED)
March 2014
By Steve Nagelstein

76% of all breaches involve stolen passwords
the security ledger
Protect yourself with Two-Factor Authentication
76% of all breaches involve stolen passwords
Homeland Security: Hack Attempts On Energy, Manufacturing Way Up in 2013
Attended cyber attacks on critical infrastructure in the U.S., including energy and critical manufacturing jumped sharply in the first half of 2013, according to a just released report from the Department of Homeland Security's Industrial Control System Cyber Emergency Response Team (ICS-CERT).
ICS-CERT said the cyber incidents across all critical infrastructure in the U.S. are on pace to double in 2013. The agency has responded to six such incidents so far. In 2012, the agency received 10 incidents for the first half of that year. Most of those incidents were in the energy sector, ICS-CERT reported.
The report is just the latest from DHS about increased energy firms after seeing a sharp jump in attacks that resulted in several 40% of the malicious activity directed at critical infrastructure reported a "wide variety of threats ranging from advanced malware and ransomware based in the US environment." Other incidents in the water and environmental sector failed to widely report on default configurations.
The report also notes that, according to 41% of the total incidents of those incidents involved attacker techniques such as spearphishing attacks.


Computer Hacker Targets Electronic Road Signs
CBS Charlotte
A hacker targeted electronic road signs in North Carolina.
The Department of Transportation says five electronic signs that were necessary for traffic lights to work were hacked on Friday morning. The messages read "Track by Dan Hester".

U.S. regulators warn banks about rise in cyber-attacks
REUTERS
U.S. regulators warn banks about rise in cyber-attacks
A group of top U.S. regulators on Wednesday warned about the threat of rising cyber-attacks on bank websites and data.

Defense Systems Has Gone Mobile
DEFENSE SYSTEMS
Defense Systems Has Gone Mobile
These systems were focused on exfiltrating information. China is using computer network exploitation capability to support intelligence collection against the U.S. diplomatic, economic and defense industrial base sectors that support U.S. national defense programs. The information logistical could potentially be used to benefit China's defense industry, high-tech industry, logistics, and military planners building a picture of U.S. network defense networks, logistics, and related military capabilities that could be exploited during a crisis.
The following are excerpted from the Pentagon report released this week, as they relate to specific sections on Chinese cyber, electronic, warfare, ISR and space activities.
MILITARY INFORMATION OPERATIONS
Chinese writings have outlined the five key features at an operational level of a maturing Chinese information operations (IO) strategy. First, Chinese authors emphasize defense as the top priority and indicate that Computer Network Defense (CND) must be the highest priority in peacetime. Chinese doctrine suggests that "tactical counteroffensives" would only be considered if an adversary's operations could not be countered.
Second, IO is viewed as an unconventional warfare weapon, which must be coordinated with other military operations and combined with other military capabilities.



Posts are written: CWE Guidance



CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (2.9) Search by ID: Go

Presentation Filter: --None--

CWE List

- Full Dictionary View
- Development View
- Research View
- Fault Pattern View
- Reports
- Mapping & Navigation

About

- Sources
- Process
- Documents
- FAQs

Community

- Use & Citations
- SwA On-Ramp
- Discussion List
- Discussion Archives
- Contact Us

Scoring

- Prioritization
- CWSS
- CWRAF
- CWE/SANS Top 25

Compatibility

- Requirements
- Coverage Claims
- Representation
- Compatible Products
- Make a Declaration

News

- Calendar
- Free Newsletter

[Search the Site](#)

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Weakness ID: 120 (*Weakness Base*) Status: Incomplete

▼ **Description**

Description Summary

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

.....

▼ **Detection Methods**

Automated Static Analysis

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.

Automated static analysis generally does not account for environmental considerations when reporting out-of-bounds memory operations. This can make it difficult for users to determine which warnings should be investigated first. For example, an analysis tool might report buffer overflows that originate from command line arguments in a program that is not expected to run with setuid or other special privileges.

Effectiveness: High

Detection techniques for buffer-related errors are more mature than for most other weakness types.

Automated Dynamic Analysis

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Manual Analysis

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

Automated Static Analysis - Binary / Bytecode

According to SOAR, the following detection techniques may be useful:





More guidance is generated



- Home
- About OWASP
- Acknowledgements
- Advertising
- AppSec Events
- Books
- Brand Resources
- Chapters
- Donate to OWASP
- Downloads
- Funding
- Governance
- Initiatives
- Mailing Lists
- Membership
- Merchandise
- News
- Community portal
- Presentations
- Press
- Projects
- Video
- Volunteer

Page [Discussion](#)

Buffer Overflows

.....

General Prevention Techniques

A number of general techniques to prevent buffer overflows include:

- Code auditing (automated or manual)
- Developer training – bounds checking, use of unsafe functions, and group standards
- Non-executable stacks – many operating systems have at least some support for this
- Compiler tools – StackShield, StackGuard, and Libsafe, among others
- Safe functions – use `strncat` instead of `strcat`, `strncpy` instead of `strcpy`, etc
- Patches – Be sure to keep your web and application servers fully patched, and be aware of bug reports relating to applications upon which your code is dependent.
- Periodically scan your application with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications.





Agenda



- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**





Writing rules is hard

You know it when you see it

Turn up sensitivity => False positives

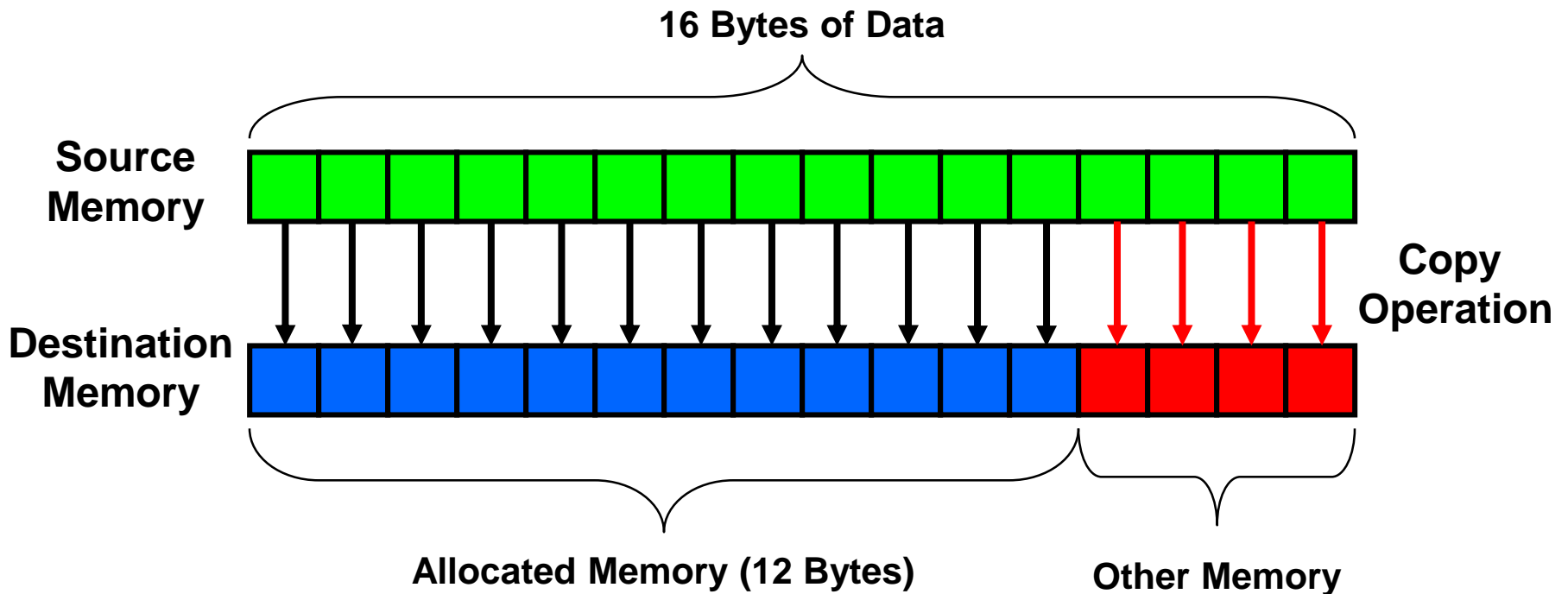
Turn up selectivity => False negatives





What Is a Buffer Overflow?

A buffer overflow occurs when data is written (or accessed) outside of the boundaries of the memory allocated to a particular data structure.





Buffer overflow: check your bounds

A programmer might code a bounds-check such as

```
char *ptr; // ptr to start of array
char *max; // ptr to end of array
size_t pos; // index input unknown to programmer
if (ptr + pos > max)
    return EINVAL;
```





Buffer overflow: check your bounds

A programmer might code a bounds-check such as

```
char *ptr; // ptr to start of array
char *max; // ptr to end of array
size_t pos; // index input unknown to programmer
if (ptr + pos > max)
    return EINVAL;
```

If **pos** is very large, it can cause **ptr + pos** to overflow, which typically wraps around — pointing to an address that is actually *lower* in memory than **ptr** (and **max**).





Buffer overflow: check your bounds

A programmer might code a bounds-check such as

```
char *ptr; // ptr to start of array
char *max; // ptr to end of array
size_t pos; // index input unknown to programmer
if (ptr + pos > max)
    return EINVAL;
```

If **pos** is very large, it can cause **ptr + pos** to overflow, which typically wraps around — pointing to an address that is actually *lower* in memory than **ptr** (and **max**).

Since (overflowed) **ptr + pos** is less than **max**, execution proceeds





Buffer overflow: surprising code elimination

One might write a check like this:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```





Buffer overflow: surprising code elimination

One might write a check like this:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

A compiler could assume that

- Programs are well defined





Buffer overflow: surprising code elimination

One might write a check like this:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

A compiler could assume that

- Programs are well defined
- Hence `ptr + len` will not overflow





Buffer overflow: surprising code elimination

One might write a check like this:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

A compiler could assume that

- Programs are well defined
- Hence `ptr + len` will not overflow
- Hence, since `len` is unsigned, `ptr + len` must be greater than or equal to (not less than) `ptr`





Buffer overflow: surprising code elimination

One might write a check like this:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

A compiler could assume that

- Programs are well defined
- Hence `ptr + len` will not overflow
- Hence, since `len` is unsigned, `ptr + len` must be greater than or equal to (not less than) `ptr`
- Hence `ptr + len < ptr` is always true and can be removed as dead code





Buffer overflow: surprising optimization

In our example:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

This optimization proceeds as follows:

```
ptr + len < ptr
```





Buffer overflow: surprising optimization

In our example:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

This optimization proceeds as follows:

```
ptr + len < ptr
ptr + len < ptr + 0
```





Buffer overflow: surprising optimization

In our example:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

This optimization proceeds as follows:

```
ptr + len < ptr
```

```
ptr + len < ptr + 0
```

```
ptr + len < ptr + 0
```




Buffer overflow: surprising optimization

In our example:

```
if (ptr + len < ptr || ptr + len > max)
    return EINVAL;
```

This optimization proceeds as follows:

```
ptr + len < ptr
ptr + len < ptr + 0
ptr + len < ptr + 0
len < 0 (impossible, len is unsigned)
```

The rewritten `len < 0` is removed.





Mitigation

This problem is easy to remediate, once it is called to the attention of the programmer, such as by a diagnostic message when dead code is eliminated.

For example, if `ptr` is less-or-equal-to `max`, then the programmer could write:

```
if (len > max - ptr)
    return EINVAL;
```

This conditional expression eliminates the possibility of undefined behavior.





Agenda



- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**





An alternative methodology for rule creation

Exploit language ambiguities

Analyze vulnerable programs

Systematically test the rules

And still consult with experts





Examine language definitions and standards for undefined, unspecified and implementation-defined behavior

3.4.3

1 **undefined behavior**

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

2 **NOTE** Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 **EXAMPLE** An example of undefined behavior is the behavior on integer overflow.

3.4.4

1 **unspecified behavior**

use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance

2 **EXAMPLE** An example of unspecified behavior is the order in which the arguments to a function are evaluated.

Source: <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf> (ISO 9899 - Programming Languages – C draft)





Examine vulnerable code for patterns

Malware repository with millions of unique, tagged artifacts

CERT Secure Coding Team has evaluated over 100M LOC



Vulnerability Notes Database

Advisory and mitigation information about software vulnerabilities

CERT Knowledgebase

The CERT Knowledgebase is a collection of internet security information related to incidents and vulnerabilities. The CERT Knowledgebase houses the public [Vulnerability Notes Database](#) as well as two restricted-access components:

- [Vulnerability Card Catalog](#) contains descriptive and referential information regarding thousands of vulnerabilities reported to the CERT Coordination Center.
- [Special Communications Database](#) contains briefs that provide advance warning and important information about vulnerabilities, intruder activity, or other critical security threats.



Implement candidate rules and run against sample code

- Focus rule when possible to
 - maximize true positive of weakness (tag bad code)
 - minimize false negative of weakness (don't tag good code)
- Write program to evaluate source code for particular rule
- Run program against collection of known bad source code and a collection of other (suspected good) code to check sensitivity and specificity of results





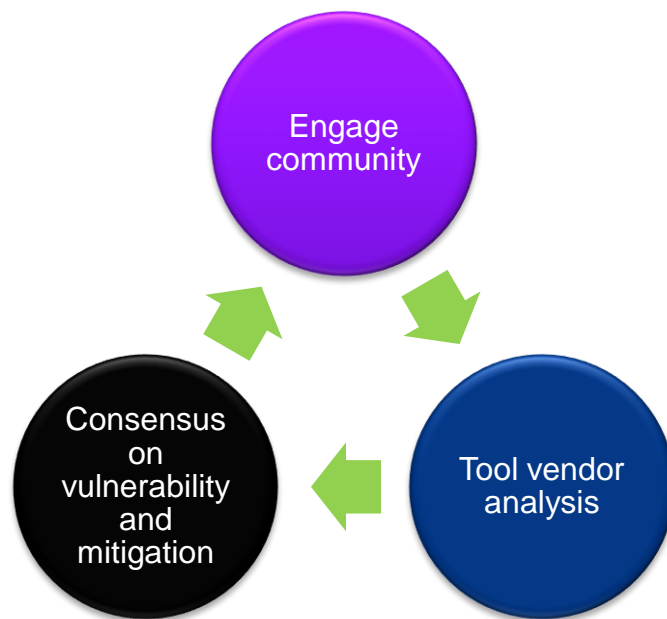
Experience with systematic testing

- Candidate rule typical evaluation
 - 10 iterations of proposed rule and associated checker
 - 7 internal evaluations
 - 3 external evaluations
- Each evaluation iteration carried out against > 10M lines of representative code
 - Variety of domains
 - Variety of code quality
- As part of creating C++ standard, general methodology applied to generate 46 rules and corresponding Clang C++ checkers
 - 19 by CERT researchers, 27 by others





Tapping into expert knowledge for developing CERT coding standards





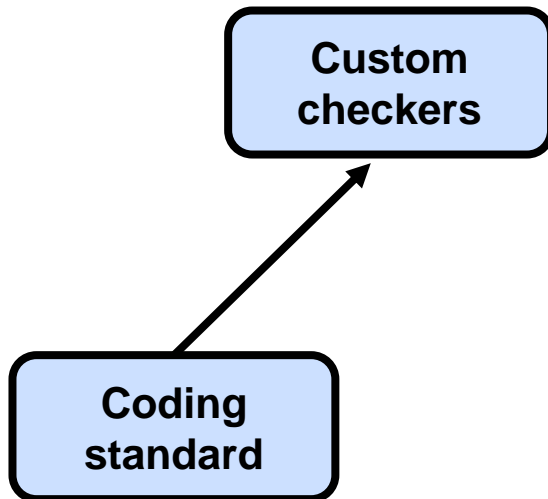
Evolution of coding support

**Coding
standard**



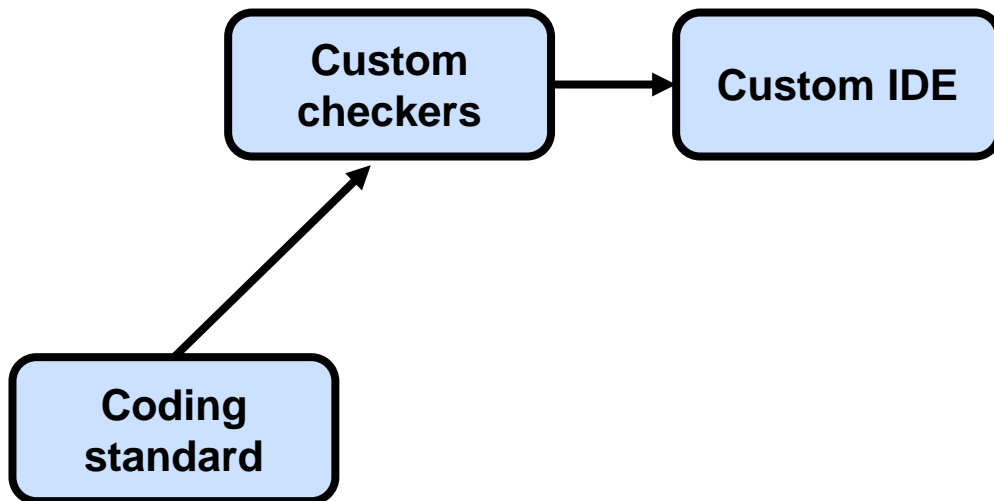


Evolution of coding support



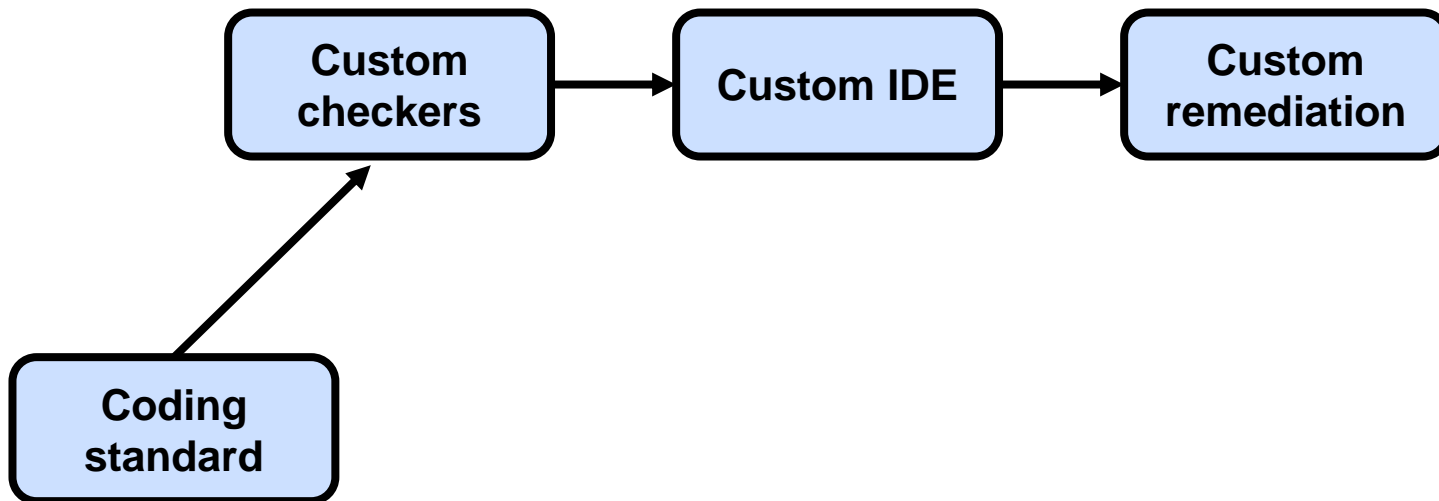


Evolution of coding support



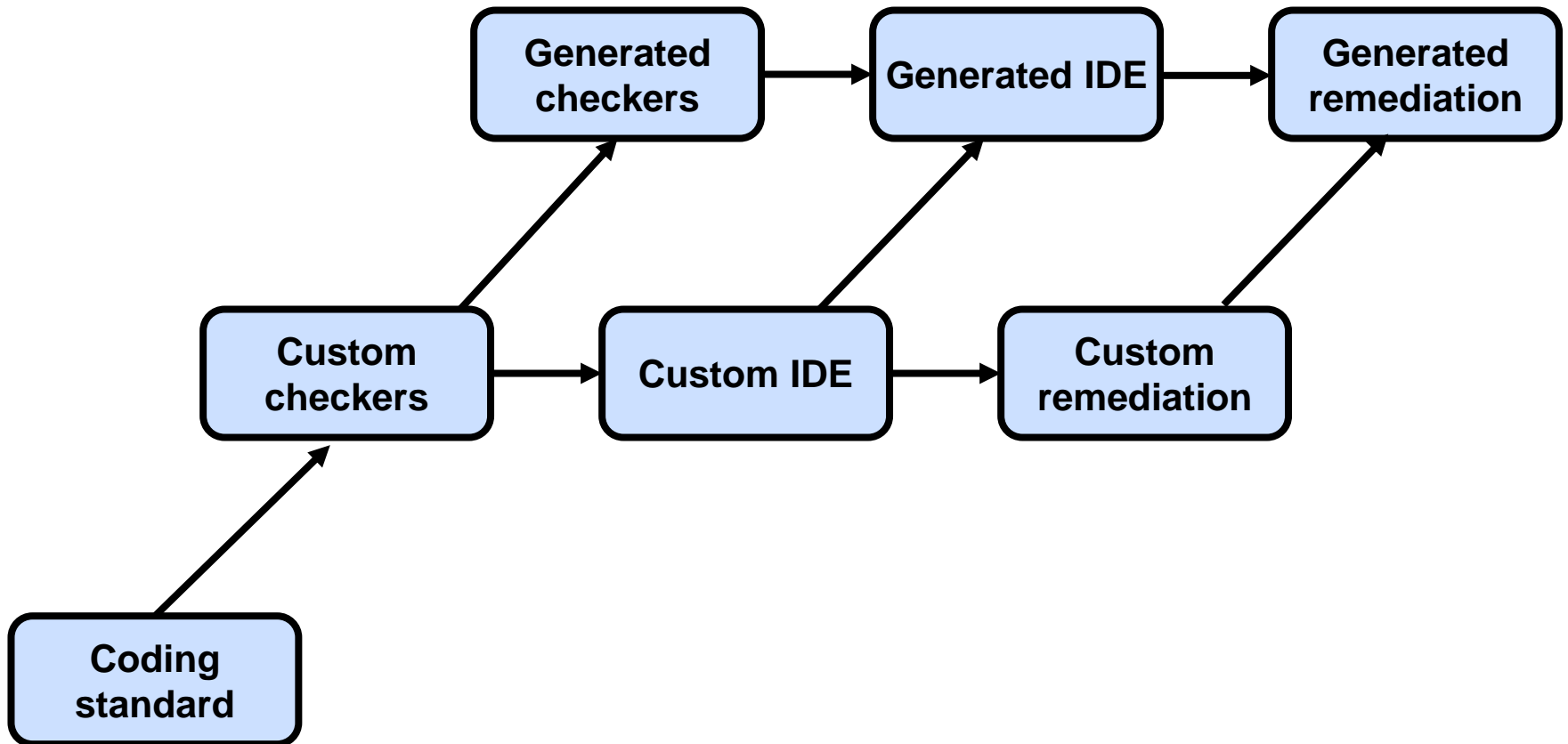


Evolution of coding support



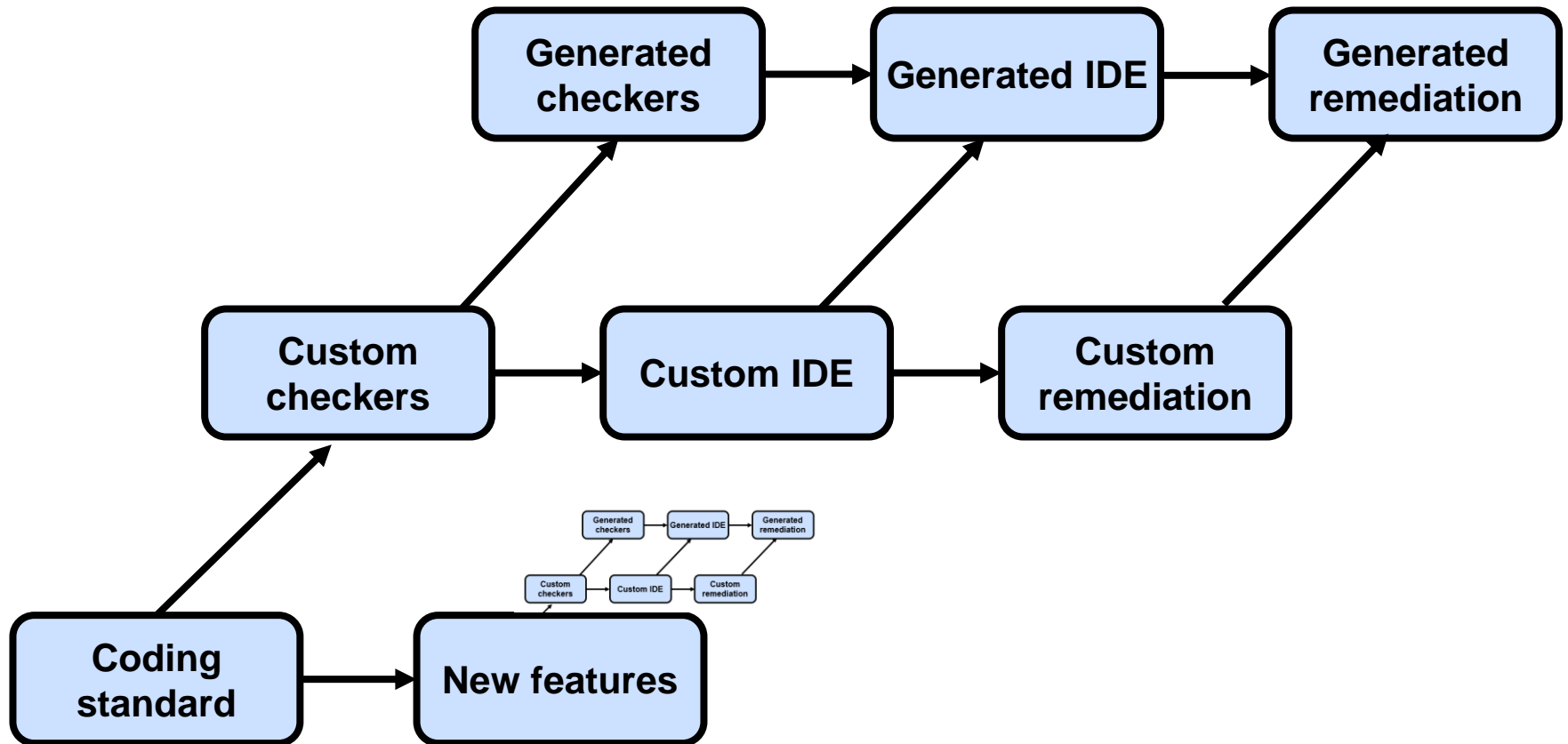


Evolution of coding support



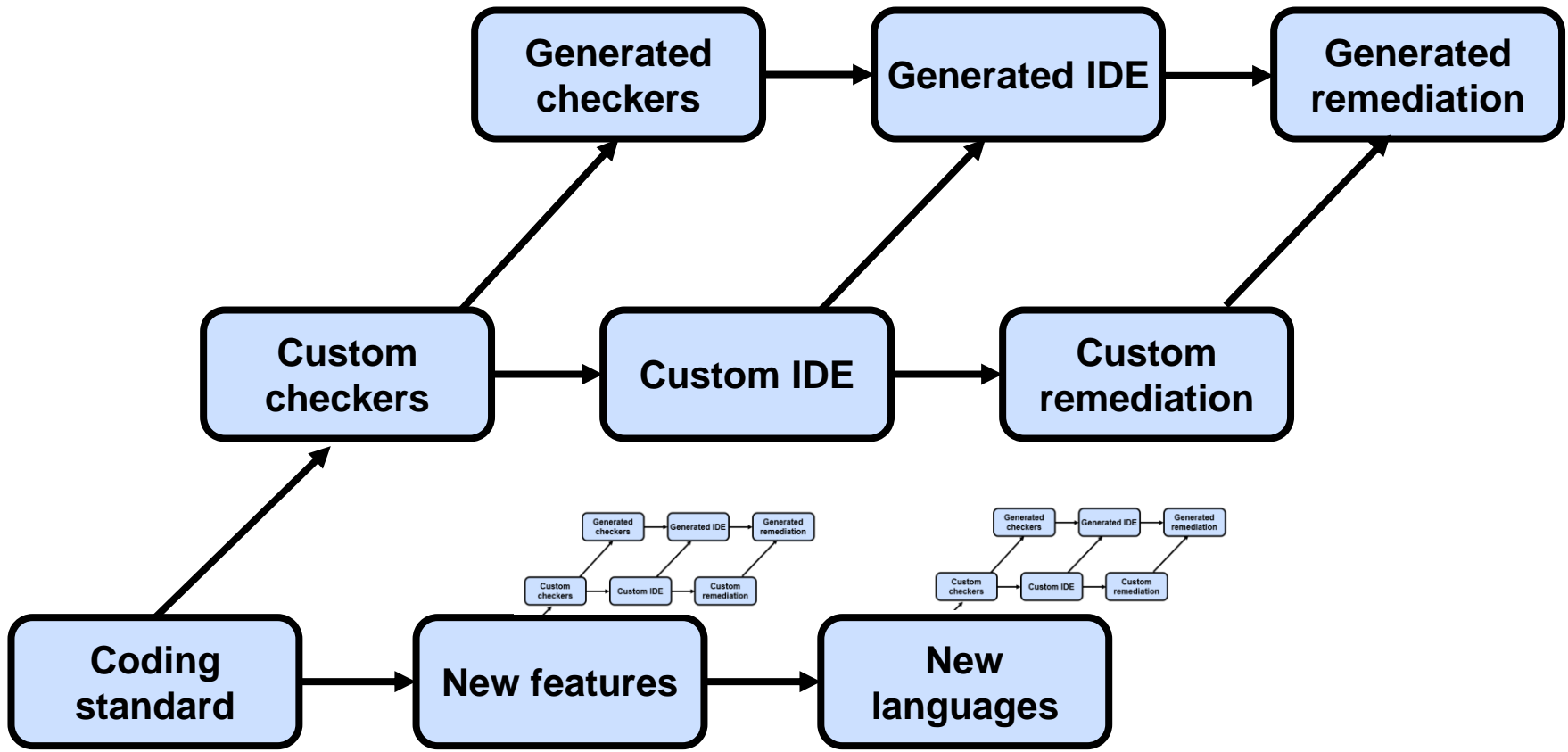


Evolution of coding support



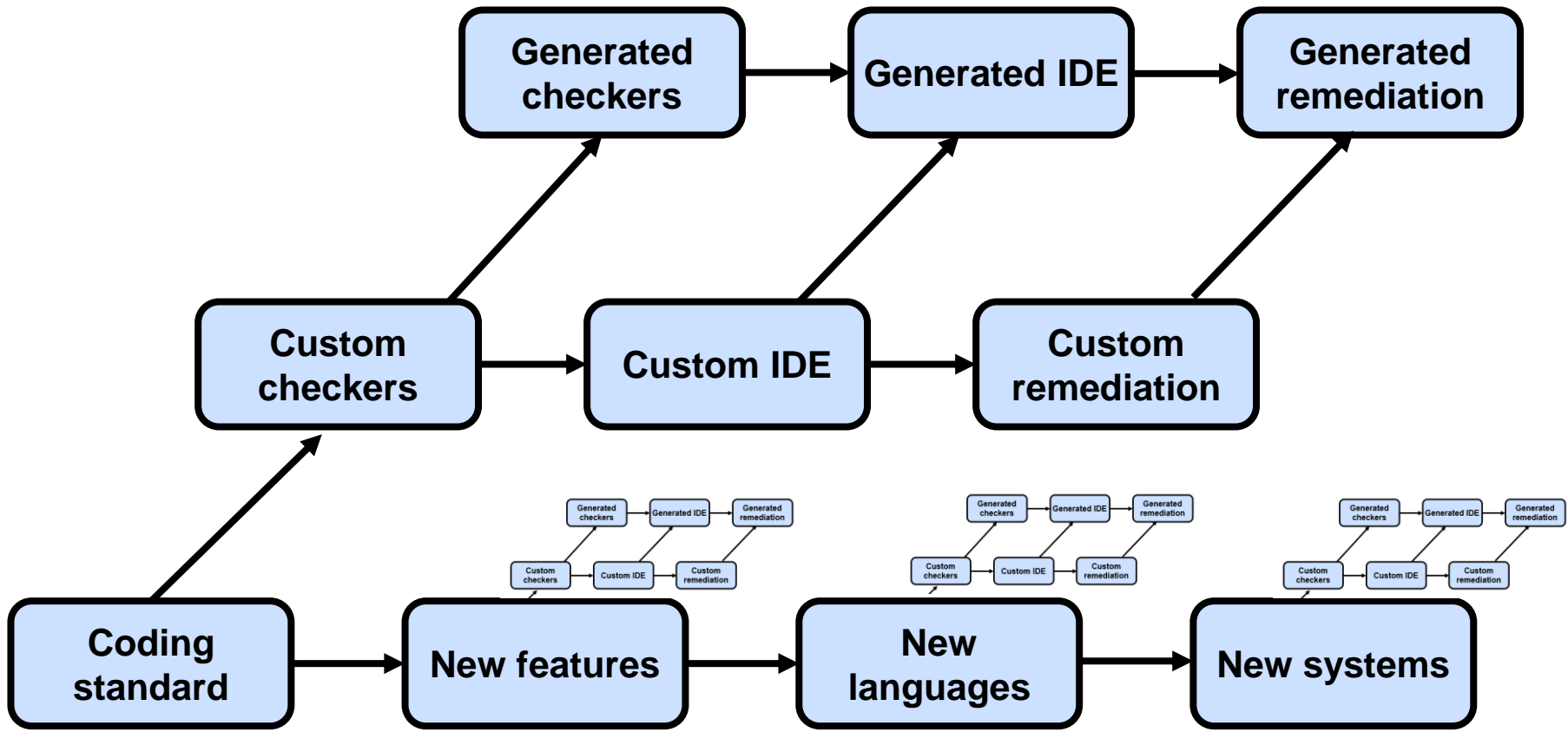


Evolution of coding support





Evolution of coding support





Agenda



- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**





CERT Secure Coding Standards

CERT C Secure Coding Standard

- Version 1.0 (C99) published in 2009
- Version 2.0 (C11) published in 2014
- ISO/IEC TS 17961 C Secure Coding Rules Technical Specification
- Conformance Test Suite

CERT C++ Secure Coding Standard

- Version 1.0 under development

CERT Oracle Secure Coding Standard for Java

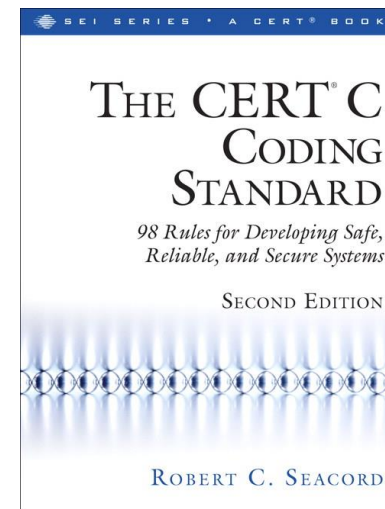
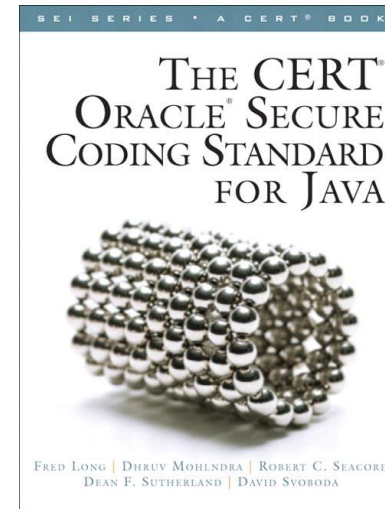
- Version 1.0 (Java 7) published in 2011
- Java Secure Coding Guidelines
- Subset applicable to Android development
- Android Annex

The CERT Perl Secure Coding Standard

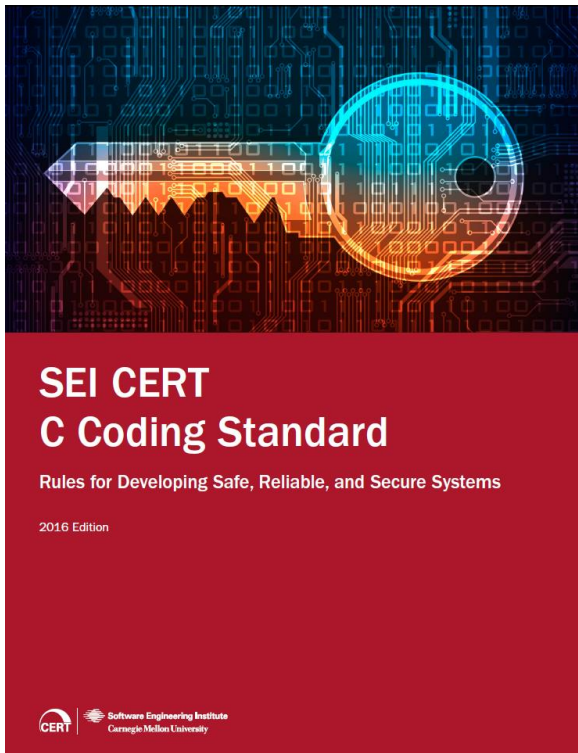
- Version 1.0 under development

CERT Python Secure Coding Standard

- Version 1.0 under development



Coding rules – 2016 Edition



- Collected wisdom of programmers and tools vendors
 - Fed by community wiki started in Spring 2006
 - Over 1,500 registered contributors
- Available as downloadable eBook
<http://cert.org/secure-coding/products-services/secure-coding-download.cfm>





Learning from rules and recommendations

Rules and recommendations in the secure coding standards focus to improve behavior

The “Ah ha” moment:
Noncompliant code examples or antipatterns in a pink frame—do not copy and paste into your code

Noncompliant Code Example

In this example, the `FormatMessage()` function allocates a buffer and stores it in the `buf` parameter. From the documentation of `FORMAT_MESSAGE_ALLOCATE_BUFFER` [MSDN]

The function allocates a buffer large enough to hold the formatted message, and places a pointer to the allocated buffer at the address specified by `lpBuffer`. The `lpBuffer` parameter is a pointer to an `LPTSTR`; you must cast the pointer to an `LPTSTR` (for example, `(LPTSTR)&lpBuffer`). The `nSize` parameter specifies the minimum number of `TCHARs` to allocate for an output message buffer. The caller should use the `LocalFree` function to free the buffer when it is no longer needed.

Instead of freeing the memory using `LocalFree()`, this code example uses `GlobalFree()` erroneously.

```
LPTSTR buf;
DWORD n = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS, 0, GetLastError(),
    LANG_USER_DEFAULT, (LPTSTR)&buf, 1024, 0);

if (n != 0) {
    /* Format and display the error to the user */
    GlobalFree(buf);
}
```

Compliant Solution

The compliant solution uses the proper deallocation function as described by the documentation.

```
LPTSTR buf;
DWORD n = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS, 0, GetLastError(),
    LANG_USER_DEFAULT, (LPTSTR)&buf, 1024, 0);

if (n != 0) {
    /* Format and display the error to the user */
    LocalFree(buf);
}
```

Compliant solutions in a blue frame that conform with all rules and can be reused in your code

Secure Coding eNewsletter engages community

Secure Coding eNewsletter
CERT Software Engineering Institute Carnegie Mellon

October 2013 Edition

News
Language Standards Updates
Upcoming Events and Training
Our People
Secure Coding Resources

News

The school year is well underway, so Dr. Robert Seacord all escaped to the Java Conference in San Francisco and gave several presentations:

- Session ID: [CON6396](#) Don't Be Fooled by Java: Programming in Java, Dean Sutfin
- Session ID: [CON3122](#) Anatomy of a Vulnerability, David Svoboda
- Session ID: [TUT5599](#) The Java Security Model, David Svoboda

All three presentations were well attended. These recordings will be made available in the coming weeks.

Secure Coding eNewsletter
CERT Software Engineering Institute Carnegie Mellon

November 2013 Edition

News
Language Standards Updates
Upcoming Events and Training
Our People
Secure Coding Resources

News

We are still working hard to complete the CERT C Secure Coding Standard upgrade for C11. To do so, we need your help in reviewing submitting comments on the wiki or by email. This is the last call for comments before publication.

Although we remain focused on security, we have begun to look at other quality attributes as well. This is reflected in the title of our most recent book, *Java Coding Standards: Recommendations for Reliable and Secure Programs*, and its revision to the CERT C Secure Coding Standard, which is now *CERT C Coding Standard: 92 Rules for Developing Safe, Secure Systems*, the tentative part being the number of rules. We hope everyone enjoyed the book that we completed last year. The final manuscript was predicted 92, as the new book is quite happy with the looking forward to the new 2014. Work is continuing on the wiki, which are not included in this or other news, ISO/IEC JTC1 SC22 WG2, their Environmentally Conscious Technology - Programming Languages, *Their Environmentally Conscious Technology - Programming Languages, Their Environmentally Conscious Technology - Programming Languages* (ISO/IEC TS 17961:2013) was officially available for purchase at the ISO store ([http://www.iso.org/number=61124](http://www.iso.org/iso/number=61124)). The purpose of ISO/IEC TS 17961 is to provide requirements for analyzers, including static analysis tools and

Secure Coding eNewsletter
CERT Software Engineering Institute Carnegie Mellon University

December 2013/January 2014

News
Language Standards Updates
Upcoming Events and Training
Our People
Secure Coding Resources

News

February / March 2014

News
Language Standards Updates
Upcoming Events and Training
Secure Coding Resources

News

It is beginning to feel like spring here in Pittsburgh: the temperature has not fallen below zero degrees for several days now, and it has even briefly stopped snowing. Many of the updates to the secure coding wiki have been in the CERT C Coding Standard space as Carol Lallier synchronizes changes from the manuscript of the upcoming [Addison Wesley book](#). This project is nearing completion-we are currently reviewing the page proofs, which are due back to the publisher on March 7. Overall, the project is on schedule and the books are still expected to be available on or about April 10, 2014.

The SEI has launched a new version of the [CERT website](#). The site has been redesigned to improve the user experience, to better represent the key capabilities and current research functions of the SEI's CERT Division, and to enable one-click access to the site's most in-demand resources (such as [Secure Coding](#)).

Signup: info@sei.cmu.edu.



Software Engineering Institute

Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.



Agenda



- **Need for secure coding standards**
- **Common rule development methodology**
- **Creating rules is difficult**
- **Systematic rule development**
- **CERT Coding Standards**
- **Summary**





Summary

Application of coding standards can avoid many vulnerabilities seen in the field

Making a standard should be based on more than opinion

Prescriptive standards give

- Developers actionable guidance to create secure code
- Tool makers actionable guidance to create testers for secure code
- Acquirers actionable requirements for licensed or developed code





Contact Information

Mark Sherman

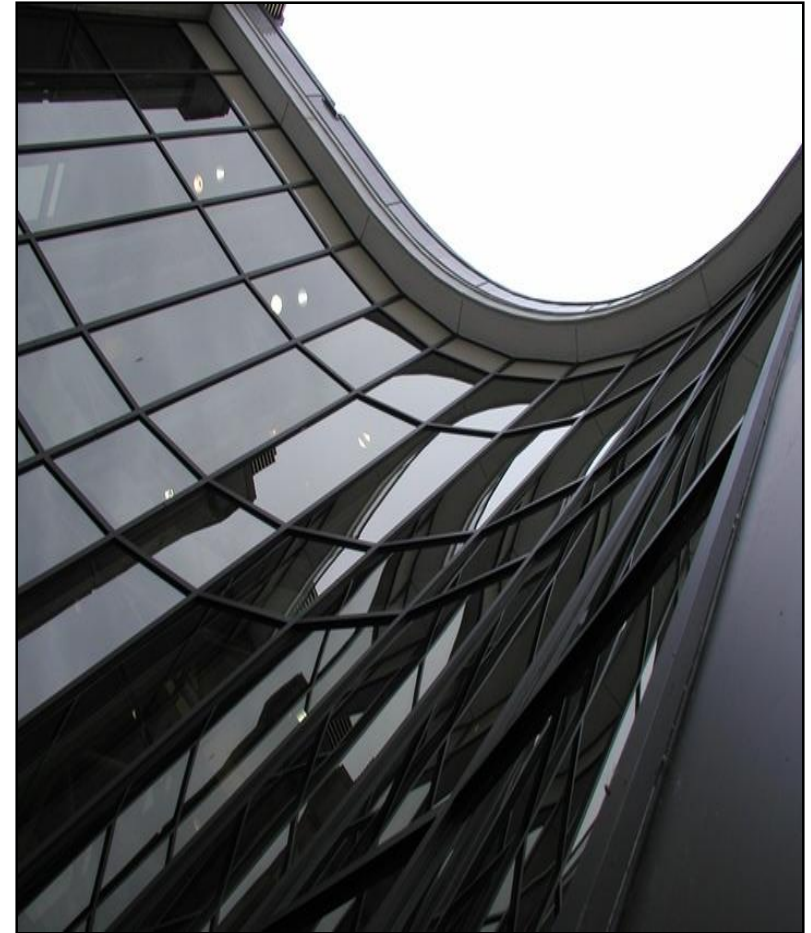
(412) 268-9223

mssherman@sei.cmu.edu

Web Resources (CERT/SEI)

<http://www.cert.org/>

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





Software Engineering Institute

Carnegie Mellon University



Software Engineering Institute

Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.