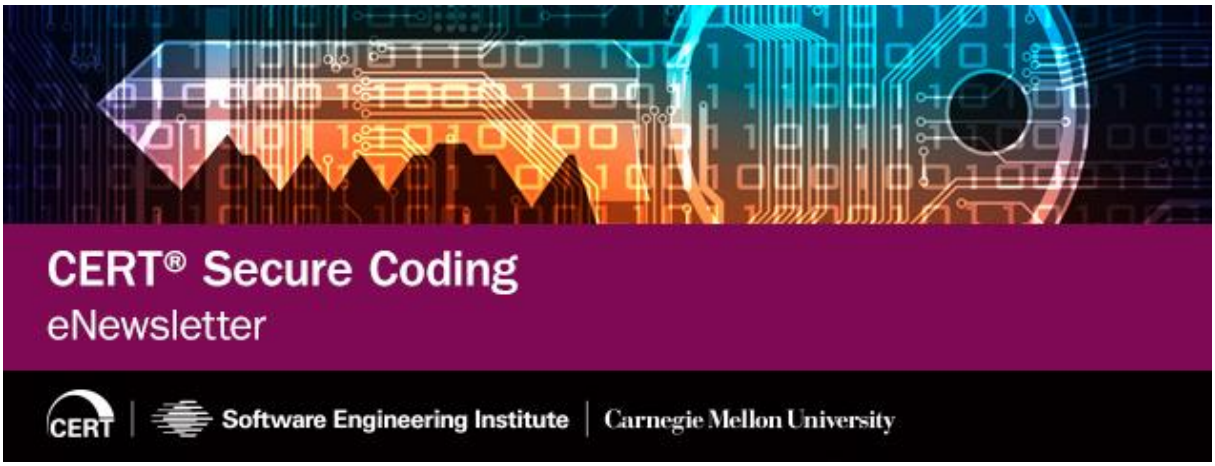

February/March 2016

Greetings

I am Bob Schiela, and I have recently accepted the honor of leading the Secure Coding Team. In the last 10 years or so, the CERT Secure Coding Initiative paved the way for defining a set of rules and recommendations in an attempt to eliminate security vulnerabilities in software that are caused by improper or imprecise language use. The Secure Coding Team also transitioned this knowledge through community engagement, standards development, books, the wiki site, training, tools, and services.

I look forward to working with the Secure Coding Team and community in a new phase of secure development and software assurance. We are continuing our community engagement and technology transition and also adding new ways to help improve the state of the practice. We recently announced a Secure Coding certificate program. We also are working on future releases of the coding standards and on exciting research to improve the efficiency and effectiveness of analysis tools and to automatically repair, not just detect, defective code that could lead to vulnerabilities. We look to improve how we are measuring secure coding and its effects on improving software assurance.

My bio appears in the "Our People" section of this month's newsletter. Feel free to contact me about your thoughts and ideas through my staff profile page. I would be glad to hear from you.

Bob

News

Spring is in the air early this year. We look forward to the green grass, flowers, trees, and birds.

As part of spring housecleaning, we are modifying the identifiers we use for exceptions to CERT rules. The exception identifiers are receiving the language suffix from the rule IDs. For example,

both guidelines DCL06-C and DCL06-CPP have exceptions named DCL06-EX1. To resolve this ambiguity, these exceptions are now called DCL06-C-EX1 and DCL06-CPP-EX1.

Secure Coding Certificates

To increase security and reduce software vulnerabilities, developers need training on how to find and fix coding errors early in the development lifecycle. Consequently, the SEI now offers the CERT Secure Coding Professional Certificate - C & C++. This certificate is the first of two new Secure Coding Professional Certificates that provide software developers with practical instruction to meet this developer need. Both certificates are based on CERT Secure Coding Standards. Each certificate program is composed of an eLearning course and an online certificate exam. Students who complete the program earn an official certificate from CERT in recognition of their accomplishment.

Language Standards Updates

CERT C Coding Standard

Editors: Martin Sebor (Red Hat, Inc.) and Aaron Ballman (SEI/CERT)

Changed

- EXP36-C. Do not cast pointers into more strictly aligned pointer types
  Added a new exception (EXP36-EX2) that permits typecasts in cases where the pointer is known to be correctly aligned by the target type. (This exception was already implicit in the rule's compliant solutions.)
- MSC32-C. Properly seed pseudorandom number generators
  Added text about system clock resolution, removed outdated warning.
- MSC37-C. Ensure that control never reaches the end of a non-void function
  Added a missing include to a compliant solution, updated implementation defined behavior wording for GCC 5.3.
- MEM30-C. Do not access freed memory
  Corrected CVE-2009-1364 code examples so that they would compile.
- INT35-C. Use correct integer precisions
  Clarified how left-shifting into padding bits can result in undefined behavior.
- INT32-C. Ensure that operations on signed integers do not result in overflow
  Added functions from `<ctype.h>` to the list of functions that take an `int` but treat the values as a `char`.
- FIO45-C. Avoid TOCTOU race conditions while accessing files
  Reinstated text that was accidentally removed regarding the "x" mode for fopen().
- CON30-C. Clean up thread-specific storage
  Removed a statement that thread-specific storage destructors were deliberately underspecified. They were intended to work like POSIX and WG 14 decided that they do.
- CON31-C. Do not destroy a mutex while it is locked and BB. Definitions
  Corrected a misunderstanding of critical sections. They are code that accesses shared data, not the shared data itself.

- CON32-C. Prevent data races when accessing bit-fields from multiple threads
  Clarified text to avoid implying that C11 changed the ABI for small adjacent non-bit-field struct members. Changing one such member is now guaranteed not to change an adjacent one, but the storage layout is in all known cases still the same.
- STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string Improved the error handling of the code examples.
- FLP34-C. Ensure that floating-point conversions are within range of the new type
  Corrected second compliant solution to properly handle negative inputs. Also, changed the first CS to check the width of the integer before storing into it. Previously, the solution had relied on implementation-defined and potentially undefined behavior to detect undefined behavior.
- ENV33-C. Do not call system() Added missing `#include` directives and corrected code examples so that they would compile. Clarified normative wording, added information about `_popen()` on Windows. Added exception to permit calling `system()` with a null argument to determine the existence of a command interpreter.
- ARR32-C. Ensure size arguments for variable length arrays are in a valid range
  Corrected an NCCE to handle memory allocation failure, since that was not the concept being demonstrated by the NCCE.
- SIG31-C. Do not access shared objects in signal handlers
  Corrected the lock-free CS to use the proper macro checking and fixed a simple typo.
- CON34-C. Declare objects shared between threads with appropriate storage durations
  In the second NCCE and its first CS, removed code that attempted to check a return value from `tss_delete()`, which does not return a value.
- CON37-C. Do not call signal() in a multithreaded program
  Changed the CS to use an `atomic_bool`. The atomic_flag was being used incorrectly and was less suitable.
- CON38-C. Preserve thread safety and liveness when using condition variables
  Corrected the `printf()` conversion specifiers to match the size of the argument to be printed.
- CON40-C. Do not copy pthread synchronization objects
  Corrected the `compute_sum()` function to return a value and added the proper initialization for the atomic variables.
- FLP36-C. Preserve precision when converting integral values to floating-point type
  Corrected the computation of the number of bits of precision in a floating-point number.
- SIG30-C. Call only asynchronous-safe functions within signal handlers
  Changed calls to `fprintf()` to use `fputs()` instead because there were no arguments beyond the format string.
- SIG34-C. Do not call signal() from within interruptible signal handlers Restored the statement that there is no safe way to implement persistent signal handling in Windows. Previously, this rule had been modified to indicate that it could be done safely. However, this was based on a misunderstanding of where the race window is. This time, an explanation was added to make it clear why it is unsafe.
- SIG35-C. Do not return from a computational exception signal handler
  Added extra code to the NCCE and CS to check the result of the call to `strtol()` before using it.
- DCL41-C. Do not declare variables inside a switch statement before the first case label
  Added `not-for-cpp` label, because such a construct is ill-formed in C++.
- EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic
  Corrected format specifiers for `size_t` in the code examples.
- FIO20-C. Avoid unintentional truncation when using fgets() or fgetws()
  Fixed an off-by-one error in the expandable compliant solution. Also added a compliant solution that uses POSIX `getline()`.
- Fixed a large number of broken links across the C rule pages.

- Reorganized the front-matter material to more closely match the structure of *The CERT C Coding Standard, Second Edition*.

New Clang Checkers

- FLP30-C. Do not use floating-point variables as loop counters
- ENV33-C. Do not call system()

CERT C++ Secure Coding Standard

Editors: Martin Sebor (Red Hat, Inc.) and Aaron Ballman (SEI/CERT)

Added

- CON50-CPP. Do not destroy a mutex while it is locked
  Ported from CON31-C with examples that illustrate the C++ style.
- CON51-CPP. Always use a lock guard to lock a mutex
  Basic hygiene for C++ mutexes in the presence of exceptions.

Changed

- OOP11-CPP. Do not copy-initialize members or base classes from a move constructor
  Added an NCCE/CS pair for initalizing member objects.
- OOP54-CPP. Gracefully handle self-assignment
  Corrected a typo in the examples.
- CTR54-CPP. Do not subtract iterators that do not refer to the same container
  Corrected a typo in the last NCCE.

CERT Oracle Secure Coding Standard for Java

Editors: Brad Senetza (Oracle) and David Svoboda (SEI/CERT)

Changed

- FIO52-J. Do not store unencrypted sensitive information on the client side includes user names as sensitive information. Generally a user name is public, and accompanied by a sensitive password, so the user name itself need not be considered sensitive.
- ERR54-J. Use a try-with-resources statement to safely handle closeable resources
  Fixed the ordering of which exception is suppressed when a try-with-resources statement produces multiple exceptions.
- OBJ13-J. Ensure that references to mutable objects are not exposed
  Pointed out that the noncompliant code example also violates OBJ01-J. Limit accessibility of fields
- NUM00-J. Detect or prevent integer overflow
  Methods in the code examples no longer have throws `ArithmeticException` because it is a runtime exception.
- LCK07-J. Avoid deadlock by requesting and releasing locks in the same order
  Made static variable `nextID` into an `AtomicLong` in the "Ordered Locks" Compliant Solution to make the constructor thread safe.
- LCK08-J. Ensure actively held locks are released on exceptional conditions
  Lock objects are now private final class objects (instead of data objects), as mandated by LCK00-J. Use private final lock objects to synchronize classes that may interact with untrusted code.

No Java rules were removed.

CERT Secure Coding Standard for Android
Editors: Fred Long (Aberystwyth University) and Lori Flynn (SEI/CERT)

No Android rules were added, removed, deprecated, or substantively changed.

CERT Perl Secure Coding Standard
Editor: David Svoboda (SEI/CERT)

No Perl rules were added, removed, deprecated, or substantively changed.

Upcoming Events

Conference: Lori Flynn is co-chairing the research track of MobileSoft 2016, tthe 3rd ACM International Conference on Mobile Software Engineering and Systems (May 16-17, 2016) in Austin, TX (USA). Anyone who is doing research on development, testing, and security related to mobile software is encouraged to attend. The MobileSoft conference is co-located with ICSE 2016 (May 14-22).

Our People
In the eNewsletter, we highlight staff members behind our secure coding research. This month we feature Robert (Bob) Schiela.

Bob Schiela is a Technical Manager, leading the Secure Coding Team in the Cyber Security Foundations directorate of the CERT Division of the SEI. Bob has been working in the field of information technology, software development, and software development education for almost 20 years. He has been helping to lead research teams and projects in Cyber Security Foundations for four years, primarily with the Science of Cyber Security group. Prior to joining Cyber Security Foundations, Bob was the Technical Advisor to the Director of the SEI. Before that position, Bob held positions in the Transition Strategy and Partner Network groups at the SEI. The majority of his work at the SEI has been related to software process improvement, software quality, and security.

Before coming to the SEI, Bob was a senior systems administrator for five years for an online-education spinoff from Carnegie Mellon University. He has an MBA and BS in Electrical and Computer Engineering from Carnegie Mellon University. He also has a PMP certification and a GIAC certification.

Secure Coding Resources

Listen to the Podcast - *Is Java More Secure than C?* by David Svoboda

Read The Blog post - *Empirical Evaluation of API Usability and Security* by Sam Weber

Read - The CERT Cybersecurity Training & Education Course Catalog

Subscribe to Our eNewsletter

Join the SEI CERT Secure Coding Community

Software Engineering Institute, Carnegie Mellon University | 4500 Fifth Avenue | Pittsburgh | PA | 15213