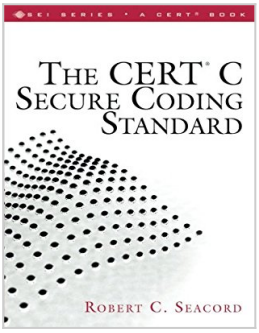

The CERT C Secure Coding Standard

By Robert C. Seacord

The CERT C Secure Coding Standard By Robert C. Seacord

"I'm an enthusiastic supporter of the CERT Secure Coding Initiative. Programmers have lots of sources of advice on correctness, clarity, maintainability, performance, and even safety. Advice on how specific language features affect security has been missing. The CERT® C Secure Coding Standard *fills this need."*
–Randy Meyers, Chairman of ANSI C

"For years we have relied upon the CERT/CC to publish advisories documenting an endless stream of security problems. Now CERT has embodied the advice of leading technical experts to give programmers and managers the practical guidance needed to avoid those problems in new applications and to help secure legacy systems. Well done!"
–Dr. Thomas Plum, founder of Plum Hall, Inc.

"Connectivity has sharply increased the need for secure, hacker-safe applications. By combining this CERT standard with other safety guidelines, customers gain all-round protection and approach the goal of zero-defect software."
–Chris Tapp, Field Applications Engineer, LDRA Ltd.

"I've found this standard to be an indispensable collection of expert information on exactly how modern software systems fail in practice. It is the perfect place to start for establishing internal secure coding guidelines. You won't find this information elsewhere, and, when it comes to software security, what you don't know is often exactly what hurts you."
–John McDonald, coauthor of *The Art of Software Security Assessment*

Software security has major implications for the operations and assets of organizations, as well as for the welfare of individuals. To create secure software, developers must know where the dangers lie. Secure programming in C can be more difficult than even many experienced programmers believe.

This book is an essential desktop reference documenting the first official release of ***The CERT® C Secure Coding Standard*** . The standard itemizes those coding

errors that are the root causes of software vulnerabilities in C and prioritizes them by severity, likelihood of exploitation, and remediation costs. Each guideline provides examples of insecure code as well as secure, alternative implementations. If uniformly applied, these guidelines will eliminate the critical coding errors that lead to buffer overflows, format string vulnerabilities, integer overflow, and other common software vulnerabilities.

The CERT C Secure Coding Standard

By Robert C. Seacord

The CERT C Secure Coding Standard By Robert C. Seacord

"I'm an enthusiastic supporter of the CERT Secure Coding Initiative. Programmers have lots of sources of advice on correctness, clarity, maintainability, performance, and even safety. Advice on how specific language features affect security has been missing. The CERT® C Secure Coding Standard *fills this need."*
–Randy Meyers, Chairman of ANSI C

"For years we have relied upon the CERT/CC to publish advisories documenting an endless stream of security problems. Now CERT has embodied the advice of leading technical experts to give programmers and managers the practical guidance needed to avoid those problems in new applications and to help secure legacy systems. Well done!"
–Dr. Thomas Plum, founder of Plum Hall, Inc.

"Connectivity has sharply increased the need for secure, hacker-safe applications. By combining this CERT standard with other safety guidelines, customers gain all-round protection and approach the goal of zero-defect software."
–Chris Tapp, Field Applications Engineer, LDRA Ltd.

"I've found this standard to be an indispensable collection of expert information on exactly how modern software systems fail in practice. It is the perfect place to start for establishing internal secure coding guidelines. You won't find this information elsewhere, and, when it comes to software security, what you don't know is often exactly what hurts you."
–John McDonald, coauthor of *The Art of Software Security Assessment*

Software security has major implications for the operations and assets of organizations, as well as for the welfare of individuals. To create secure software, developers must know where the dangers lie. Secure programming in C can be more difficult than even many experienced programmers believe.

This book is an essential desktop reference documenting the first official release of ***The CERT® C Secure Coding Standard*** . The standard itemizes those coding errors that are the root causes of software vulnerabilities in C and prioritizes them by severity, likelihood of exploitation, and remediation costs. Each guideline provides examples of insecure code as well as secure, alternative implementations. If uniformly applied, these guidelines will eliminate the critical coding errors that lead to buffer overflows, format string vulnerabilities, integer overflow, and other common software vulnerabilities.

The CERT C Secure Coding Standard By Robert C. Seacord Bibliography

- Sales Rank: #1778636 in Books
- Published on: 2008-10-24

- Released on: 2008-10-14
- Original language: English
- Number of items: 1
- Dimensions: 8.90" h x 1.70" w x 7.00" l, 2.27 pounds
- Binding: Paperback
- 720 pages

Editorial Review

From the Back Cover
""I'm an enthusiastic supporter of the CERT Secure Coding Initiative. Programmers have lots of sources of advice on correctness, clarity, maintainability, performance, and even safety. Advice on how specific language features affect security has been missing. "The CERT(R) C Secure
Coding Standard" fills this need."
"-Randy Meyers, Chairman of ANSI C"

"For years we have relied upon the CERT/CC to publish advisories documenting an endless stream of security problems. Now CERT has embodied the advice of leading technical experts to give programmers and managers the practical guidance needed to avoid those problems in new applications and to help secure legacy systems. Well done!""
-Dr. Thomas Plum, founder of Plum Hall, Inc.
""Connectivity has sharply increased the need for secure, hacker-safe applications. By combining this CERT standard with other safety guidelines, customers gain all-round protection and approach the goal of zero-defect software.""
-Chris Tapp, Field Applications Engineer, LDRA Ltd.
""I've found this standard to be an indispensable collection of expert information on exactly how modern software systems fail in practice. It is the perfect place to start for establishing internal secure coding guidelines. You won't find this information elsewhere, and, when it comes to software security, what you don't know is often exactly what hurts you.""
-John McDonald, coauthor of "The Art of Software Security Assessment"

Software security has major implications for the operations and assets of organizations, as well as for the welfare of individuals. To create secure software, developers must know where the dangers lie. Secure programming in C can be more difficult than even many experienced programmers believe.
This book is an essential desktop reference documenting the first official release of "The CERT(R) C Secure Coding Standard." The standard itemizes those coding errors that are the root causes of software vulnerabilities in C and prioritizes them by severity, likelihood of exploitation, and remediation costs. Each guideline provides examples of insecure code as well as secure, alternative implementations. If uniformly applied, these guidelines will eliminate the critical coding errors that lead to buffer overflows, format string vulnerabilities, integer overflow, and other common software vulnerabilities.

About the Author
Robert C. Seacord leads the Secure Coding Initiative at the CERT at the Software Engineering Institute (SEI) in Pittsburgh, Pennsylvania. The CERT, among other security-related activities, regularly analyzes software vulnerability reports and assesses the risk to the Internet and other critical infrastructure. Robert is an adjunct professor in the Carnegie Mellon University School of Computer Science and in the Information Networking Institute and part-time faculty at the University of Pittsburgh. An eclectic technologist, Robert is author of three previous books, *Secure Coding in C and C++* (Addison- Wesley, 2005), *Building Systems from Commercial Components* (Addison-Wesley, 2002), and *Modernizing Legacy Systems* (Addison-Wesley, 2003), as well as more than 40 papers on software security, componentbased software engineering, Web-based system design, legacy-system modernization, component repositories and search engines, and user interface design and development. Robert started programming professionally for IBM in 1982, working in communications and operating system software, processor development, and software engineering. Robert also has worked at the X Consortium, where he developed and maintained code for the Common Desktop

Environment and the X Window System. He represents Carnegie Mellon at PL22. 11 (ANSI “C”) and is a technical expert for the JTC1/SC22/WG14 international standardization working group for the C programming language.

An essential element of secure coding in the C programming language is well-documented and enforceable coding standards. Coding standards encourage programmers to follow a uniform set of guidelines determined by the requirements of the project and organization, rather than by the programmer's familiarity or preference. Once established, these standards can be used as a metric to evaluate source code (using manual or automated processes).

The CERT C Secure Coding Standard provides guidelines for secure coding in the C programming language. The goal of these guidelines is to eliminate insecure coding practices and undefined behaviors that can lead to exploitable vulnerabilities. The application of the secure coding standard will lead to higher-quality systems that are robust and more resistant to attack.

The CERT C Secure Coding Standard was developed over a period of two and a half years as a community effort and involved the efforts of 226 contributors and reviewers including a half-dozen active members of the ISO/IEC WG14 international standardization working group for the programming language C, the Chairman and Vice Chairman of PL22.11 (ANSI "C"), representatives from the Open Group, USENIX, Microsoft, and numerous other companies and organizations. Drafts of *The CERT C Secure Coding Standard* were twice reviewed by ISO/IEC WG14 and subjected to the scrutiny of the public including members of the Association of C and C++ Users (ACCU) and the `comp.lang.c` news group.

The results of this effort are 89 rules and 132 recommendations for secure coding in the C programming language. Most of these guidelines come complete with insecure (non-compliant) code examples, and secure (compliant solutions). The CERT C Secure Coding Standards are supported by training available from the Software Engineering Institute and other licensed partners. A number of source code analysis tools are available to automatically detect violations of CERT Secure Coding Standard rules and recommendations, including Compass/ROSE which is freely available from Lawrence Livermore National Laboratory and CERT.

The Demand for Secure Software

The Morris worm incident, which brought ten percent of Internet systems to a halt in November 1988, resulted in a new and acute awareness of the need for secure software systems. Twenty years later, many security analysts, software developers, software users, and policy makers are asking the question "Why isn't software more secure?"

The first problem is that the term *software security*, as it is used today, is meaningless. I have attempted to define this term, as have others, but there is no generally accepted definition. Why does this matter?

There are a variety of reasons given for why software is not more secure, such as the tools are inadequate, programmers lack sufficient training, and schedules are too short. But these are all solvable problems. The root cause of the issue lies elsewhere.

The reason more software is not more secure is because there is no *demand* for secure software. In simple terms, if one vendor offers a product that has more features, better performance, and is available today and another vendor offers a *secure* product that has less features, not quite as good performance, and will be

available in six months, there is really no question as to which product customers will buy, and vendors *know* this.

So why don't customers buy secure products? Again, it is because the word "secure" is meaningless in this context. Why would a customer pass up tangible benefits to buy a product that has an ill-defined and intangible property?

This is the problem addressed by the CERT C Secure Coding Standard. This book contains 89 rules and 132 recommendations for producing secure code. While the application of these rules and recommendations does not guarantee the security of a software system, it does tell you a great deal about the quality and security of the code. It tells you that the software was developed to a set of industry standard rules and recommendations that were developed by the leading experts in the field. It tells you that a tremendous amount of time and effort went into producing code that is free from the common coding errors that have resulted in numerous vulnerabilities that have been reported to and published by the CERT Coordination Center over the past two decades. It tells you that the software developers who produced the code have done so with a real knowledge of the types of vulnerabilities that can exist and the exploits that can be used against them, and consequently have developed the software with a real security mindset.

So, the *small* problem we have set out to address in this book is to change the market dynamic for developing and purchasing software systems. By producing an *actionable* definition of software security for C language programs--compliance with the rules and recommendations in this standard--we have defined a mechanism by which customers can demand secure software systems and vendors can comply. Furthermore, the concept of a secure system now has *value* because the word "secure" has meaning.

History

I have participated in C language standardization efforts for the past several years as the Carnegie Mellon University representative to INCITS J11 (now PL22.11) and as a technical expert at ISO/IEC WG14 (the international standardization working group for the programming language C). The first WG14 meeting I attended was held in April 2005 in Lillehammer, Norway, where we discussed a proposal for a Specification for Secure C Library Functions. That specification would eventually be published as TR 24731-1, *Extensions to the C Library - Part 1: Bounds Checking Interfaces* .

Although the TR 24731-1 are "more secure," they are still susceptible to reuse. Consequently, a separate effort was started at the WG14 meeting to develop PDTR 24731-2, *Extensions to the C Library - Part II: Dynamic Allocation Functions* ISO/IEC PDTR 24731-2, consisting primarily of existing POSIX and Linux functions. At that time, I thought it would make sense to develop a managed string library to provide a set of dynamic allocation functions with a consistent API and propose it to WG14 for standardization. One year later in Berlin, Germany, I was told that I had come up with a good technical solution but there was no customer demand for such a library.

Minutes later, during a break, I had a "Mean Joe Green" moment in the hallway when Tom Plum approach me and suggested that perhaps the C programming community would benefit from CERT developing a secure coding standard. I immediately saw the wisdom of this proposal. The C99 standard is a authoritative document, but the audience for it is primarily compiler implementers and, as been noted by many, its language is obscure and often impenetrable. A secure coding standard would be primarily targeted towards C language programmers and would provide actionable guidance on how to code securely in the language.

Community Development Process

The development of a secure coding standard for any programming language is a difficult undertaking that requires significant community involvement. The following development process has been used to create this standard:

1. Rules and recommendations for a coding standard are solicited from the communities involved in the development and application of each programming language, including the formal or *de facto* standard bodies responsible for the documented standard and user groups.
2. These rules and recommendations are edited by members of the CERT technical staff and industry experts for content and style on the CERT Secure Coding Standards wiki at www.securecoding.cert.org.
3. The user community reviews and comments on the publicly posted content using threaded discussions and other communication tools. If a consensus develops that the rule or recommendation is appropriate and correct, it is incorporated into an officially released version of the secure coding standard. If the rule does not achieve consensus, it is moved to a special section called "The Void". From here, it may be resurrected (usually in a altered form) or removed.

This development approach has been highly successful with numerous individuals and organizations contributing their time and expertise to the project. As a result of this process, *The CERT C Secure Coding Standard* has achieved a level of completeness and thoroughness that would not otherwise be achievable by a single author, or even a small team of authors.

The main disadvantage of developing a secure coding standard on a wiki is that the content is constantly evolving. This is great if you want the latest information, and you ware willing to entertain the possibility that a recent change has not yet been fully vetted. However, many software development organizations require a final document before they can commit to complying with a (fixed) set of rules and recommendations. This book serves that purpose, as Version 1.0 of the CERT C Secure Coding Standard.

With the production of the manuscript for this book in June of 2008, Version 1.0 (this book) and the wiki versions of the Secure Coding Standard began to diverge. Because both the C programming language and our knowledge of how to use it securely is still evolving, CERT will continue to evolve the CERT C Secure Coding Standard" on the secure coding wiki. These changes may then be incorporated into future, officially released versions of this standard.

Purpose

The CERT C Secure Coding Standard provides developers with guidelines for secure coding in the C programming language. These guidelines serve a variety of purposes. First, they enumerate common errors in C language programming that can lead to software defects, security flaws, and software vulnerabilities. These are all errors for which a conforming compiler is not required by the standard to issue a fatal diagnostic. In other words, the compiler will generate an executable, frequently without warning, and the resulting code executable will contain flaws that may make it vulnerable to attack.

Second, this coding standard provides recommendations for how to produce secure code. Failure to comply with these recommendations does not necessarily mean that the software is insecure, but if followed these recommendations can be powerful tools in eliminating vulnerabilities from software.

Third, this coding standard identifies non-portable coding practices. Portability is not a strict requirement of security, but non-portable assumptions in code often result in vulnerabilities when code is ported to platforms for which these assumptions are no longer valid.

Guidelines are classified as either *rules* or *recommendations*. Guidelines are defined to be rules when all of the following conditions are met:

1. Violation of the coding practice is likely to result in a security flaw that may result in an exploitable vulnerability.
2. There is a denumerable set of conditions for which violating the coding practice is necessary to ensure correct behavior.
3. Conformance to the coding practice can be determined through automated analysis, formal methods, or manual inspection techniques.

Implementation of the secure coding rules defined in this standard are necessary (but not sufficient) to ensure the security of software systems developed in the C programming language.

Recommendations are guidelines or suggestions. Guidelines are defined to be recommendations when all of the following conditions are met:

1. Application of the coding practice is likely to improve system security.
2. One or more of the requirements necessary for a coding practice to be considered a rule cannot be met.

The set of recommendations that a particular development effort adopts depends on the security requirements of the final software product. Projects with high-security requirements can dedicate more resources to security and are consequently likely to adopt a larger set of recommendations.

To ensure that the source code conforms to this secure coding standard, it is necessary to have measures in place that check for rules violations. The most effective means of achieving this is to use one or more static analysis tools. Where a rule cannot be checked by a tool, then a manual review is required.

Scope

The CERT C Programming Language Secure Coding Standard was developed specifically for versions of the C programming language defined by

- ISO/IEC 9899:1999 Programming Languages -- C, Second Edition
- Technical corrigenda TC1, TC2, and TC3
- ISO/IEC TR 24731-1 Extensions to the C Library, Part I: Bounds-checking interfaces
- ISO/IEC WDTR 24731-2 Extensions to the C Library, Part II: Dynamic Allocation Functions

Most of the material included in this standard can also be applied to earlier versions of the C programming language.

Rules and recommendations included in this CERT C Programming Language Secure Coding Standard are designed to be operating system and platform independent. However, the best solutions to secure coding problems are often platform specific. In most cases, this standard provides appropriate compliant solutions for POSIX-compliant and Windows operating systems. In many cases, compliant solutions have also been provided for specific platforms such as Linux or OpenBSD. Occasionally, we also point out implementation-specific behaviors when these behaviors are of interest.

Rationale

A secure coding standard for the C programming language can create the highest value for the longest period of time by focusing on C99 and the relevant post-C99 technical reports. In addition, because considerably

more money and effort is devoted to developing new code than maintaining existing code, the highest return on investment comes from influencing programmers who are developing new code. Maintaining existing code is still an important concern, however.

The C standard documents existing practice where possible. That is, most features must be tested in an implementation before being included in the standard. The CERT C secure coding standard has a different purpose. When existing practice serves this purpose, that is fine, but the goal is to create a new set of best practices, and that includes introducing some concepts that are not yet widely known. To put it a different way, the CERT C secure coding guidelines are attempting to drive change rather than just document it.

For example, the C library technical report, part 1 (TR 24731-1) is gaining support, but at present is only implemented by a few vendors. It introduces functions such as `memcpy_s()`, which serve the purpose of security by adding the destination buffer size to the API. A forward-looking document could not reasonably ignore these simply because they are not yet widely implemented.

C99 is more widely implemented, but even if it were not yet, it is the direction in which the industry is moving. Developers of new C code, especially, need guidance that is usable on and makes the best use of the compilers and tools that are now being developed and are being supported into the future.

Some vendors have extensions to C, and some also have implemented only part of the C standard before stopping development. Consequently, it is not possible to back up and only discuss C95, or C90. The vendor support equation is too complicated to draw a line and say that a certain compiler supports exactly a certain standard. Whatever demarcation point is selected, different vendors are on opposite sides of it for different parts of the language. Supporting all possibilities would require testing the cross product of each compiler with each language feature. Consequently, we have selected a demarcation point that is the most recent in time, so that the rules and recommendations defined by the standard will be applicable for as long as possible. As a result of the variations in support, source-code portability is enhanced when the programmer uses only the features specified by C90. This is one of many trade-offs between security and portability inherent to C language programming.

The value of forward looking information increases with time before it starts to decrease. The value of backward-looking information starts to decrease immediately.

For all these reasons, the priority of this standard is to support new code development using C99 and the post-C99 technical reports. A close-second priority is supporting remediation of old code using C99 and the technical reports.

This standard does try to make contributions to support older compilers when these contributions can be significant and doing so does not compromise other priorities. The intent is not to capture all deviations from the standard but only a few important ones.

Issues Not Addressed

There are a number of issues not addressed by this secure coding standard.

- **Coding Style.** Coding style issues are subjective, and it has proven impossible to develop a consensus on appropriate style guidelines. Consequently, the CERT C Secure Coding standard does not require any particular coding style to be enforced but only that the user defines style guidelines and apply these guidelines consistently. The easiest way to consistently apply a coding style is with the use of a code formatting tool. Many interactive development environments (IDEs) provide such capabilities.
- **Tools.** As a federally funded research and development center (FFRDC), the SEI is not in a position to

recommend particular vendors or tools to enforce the restrictions adopted. The user of this document is free to choose tools, and vendors are encouraged to provide tools to enforce the rules.
- **Controversial Rules.** In general, the CERT secure coding standards try to avoid the inclusion of controversial rules that lack a broad consensus.

Who Should Read This Book

The CERT C Secure Coding Standard is primarily intended for developers of C language programs. While security is an important for Internet-facing systems, for example, it is also an important concern for any software component that may be included or deployed as part of a secure software system. With systems increasingly being composed of software components, or even other systems, it is difficult to identify situations in which software is guaranteed not to be used in another context, which perhaps has more stringent security requirements.

This book is also useful for C language programmers who don't realize they are interested in security as most of these guidelines have practical applications for achieving other quality attributes such as safety, reliability, dependability, robustness, availability, maintainability.

While not intended for C++ programmers, this book may also be of some value because the vast majority of issues identified for C language programs are also issues in C++ programmers, although in many cases the solutions are different.

Another group of individuals that can benefit from reading this book are the members of the ISO/IEC WG14 (the international standardization working group for the programming language C) as they consider software security requirements for the new major revision of the C language standard (C1X) currently being developed.

How This Book is Organized

This book is organized into an introductory chapter, thirteen chapters each containing rules and recommendations in a particular topic area, and an appendix containing rules and recommendations for POSIX to demonstrate how this secure coding standard can be customized for particular environments.

Chapter 1, Introduction
Chapter 2, Preprocessor (PRE)
Chapter 3, Declarations and Initialization (DCL)
Chapter 4, Expressions (EXP)
Chapter 5, Integers (INT)
Chapter 6, Floating Point (FLP)
Chapter 7, Arrays (ARR)
Chapter 8, Characters and Strings (STR)
Chapter 9, Memory Management (MEM)
Chapter 10, Input Output (FIO)
Chapter 11, Environment (ENV)
Chapter 12, Signals (SIG)
Chapter 13, Error Handling (ERR)
Chapter 14, Miscellaneous (MSC)
Appendix A, POSIX (POS)

The POSIX appendix is non-normative and not a prescriptive part of the standard.

Notes to the Reader

As noted, the CERT C Secure coding standard is organized into chapters, each containing a set of guidelines in a particular topic area. Each guideline in this standard has a unique *identifier*, which is included in the title.

Most guidelines have a consistent structure. The title of the guidelines and the introductory paragraphs define the rule or recommendation. This is typically followed by one or more pairs of *non-compliant code examples* and *compliant solutions*. Each guideline also includes a risk assessment and a list of appropriate *references* (where applicable). Guidelines will also include a table of *related vulnerabilities*, where identified.

Identifiers

These identifiers consist of three parts:

- A three-letter mnemonic representing the section of the standard
- A two-digit numeric value in the range of 00-99
- The letter "A" or "C" to indicate whether the coding practice is an advisory recommendation or a compulsory rule

The three-letter mnemonic can be used to group similar guidelines and to indicate to which category a guideline belongs.

The numeric value is used to give each guideline a unique identifier. Numeric values in the range of 00-29 are reserved for recommendations, while values in the range of 30-99 are reserved for rules.

The letter "A" or "C" in the identifier is not required to uniquely identify guideline. It is used only to provide a clear indication of whether the coding practice is an advisory recommendation or a compulsory rule.

Non-Compliant Code Exemples and Compliant Solutions

Non-compliant code examples are examples of insecure code that violate the guideline under discussion. It is important to note that these are only examples, and eliminating all occurrences of the example does not necessarily mean that your code is now compliant with the guideline.

The non-compliant code examples are typically followed by compliant solutions, which are examples of how the logic from the corresponding non-compliant code example can be coded in a secure, compliant manner. Except where noted, non-compliant code examples should only contain a violation of the rule under discussion. Compliant solutions should comply with all secure coding rules, but may on occasion fail to comply with a recommendation as noted.

Risk Assessment

Each guideline contains a risk assessment section, which attempts to quantify and qualify the risk of violating each guideline. This information is intended primarily for remediation projects to help prioritize repairs, as it is assumed that new development efforts will conform with the entire standard.

Each rule and recommendation has an assigned priority. Priorities are assigned using a metric based on Failure Mode, Effects, and Criticality Analysis (FMECA). Three values are assigned for each rule on a scale of 1 to 3 for

- Severity - how serious are the consequences of the rule being ignored
  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 could lead to an exploitable vulnerability
  1 = unlikely
  2 = probable
  3 = likely
- Remediation cost - how expensive is it to comply with the rule
  1 = high (manual detection and correction)
  2 = medium (automatic detection and manual correction)
  3 = low (automatic detection and correction)

The three values are then multiplied together for each rule. This product provides a measure that can be used in prioritizing the application of the rules. These products range from 1 to 27. Rules and recommendations with a priority in the range of 1-4 are level 3 rules, 6-9 are level 2, and 12-27 are level 1. As a result, it is possible to claim level 1, level 2, or complete compliance (level 3) with a standard by implementing all rules in a level.

Recommendations are not compulsory and are provided for information purposes only.

References

Guidelines include frequent references to the vulnerability notes in CERT's Coordination Center Vulnerability Notes Database, CWE IDs in MITRE's Common Weakness Enumeration (CWE) MITRE 07, and CVE numbers from MITRE's Common Vulnerabilities and Exposures (CVE).

You can create a unique URL to get more information on any of these topics by appending the relevant ID to the end of a fixed string. For example, to find more information about:

- VU#551436, "Mozilla Firefox SVG viewer vulnerable to integer overflow," you can append 551436 to https://www.kb.cert.org/vulnotes/id/ and enter the resulting URL in your browser: https://www.kb.cert.org/vulnotes/id/551436
- CWE ID 192, "Integer Coercion Error" you can append "192.html" to "http://cwe.mitre.org/data/definitions/" and enter the resulting URL in your browser: http://cwe.mitre.org/data/definitions/192.html
- CVE-2006-1174, you can append "CVE-2006-1174" to "http://cve.mitre.org/cgi-bin/cvename.cgi?name=" and enter the resulting URL in your browser: http://cve.mitre.org/cgi-bin/cvename.cgi?name= CVE-2006-1174

Guidelines are frequently correlated with language vulnerabilities in ISO/IEC PDTR 24772. *Information Technology -- Programming Languages -- Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use*

Related Vulnerabilities

Wherever possible, we have tried to link the rules and recommendations in this secure coding standard to violations of actual vulnerabilities published in the CERT Coordination Center Vulnerability Notes Database. New links are continually added. To find the latest list of related vulnerabilities, enter the

following URL:
https://www.kb.cert.org/vulnotes/bymetric?searchview&query=FIELD+KEYWORDS+contains+XXXNN-X
where "XXXNN-X" is the ID of the rule or recommendation for which you are searching.

These tables consist of four fields: *metric, ID, date public,* and *name*.

Vulnerability Metric

The CERT vulnerability metric value is a number between 0 and 180 that assigns an approximate severity to the vulnerability. This number considers several factors:

- Is information about the vulnerability widely available or known?
- Is the vulnerability being exploited in incidents reported to CERT or other incident response teams?
- Is the Internet infrastructure (e.g., routers, name servers, critical Internet protocols) at risk because of this vulnerability?
- How many systems on the Internet are at risk from this vulnerability?
- What is the impact of exploiting the vulnerability?
- How easy is it to exploit the vulnerability?
- What are the preconditions required to exploit the vulnerability?

Because the questions are answered with approximate values based on our own judgments and may differ significantly from one site to another, you should not rely too heavily on the metric for prioritizing response to vulnerabilities. Rather, this metric may be useful for separating the serious vulnerabilities from the larger number of less severe vulnerabilities described in the database. Because the questions are not all weighted equally, the resulting score is not linear (that is, a vulnerability with a metric of 40 is not twice as severe as one with a metric of 20).

An alternative vulnerability severity metric is the Common Vulnerability Scoring System (CVSS).

Vulnerability ID

Vulnerability ID numbers are assigned at random to uniquely identify a vulnerability. These IDs are four to six digits long, and are usually prefixed with "VU#" to mark them as vulnerability IDs.

Date Public

This is the date on which the vulnerability was first known to the public, to the best of our knowledge. Usually this date is when the Vulnerability Note was first published, when an exploit was first discovered, when the vendor first distributed a patch publicly, or when a description of the vulnerability was posted to a public mailing list. By default, this date is set to be our Vulnerability Note publication date.

Vulnerability Name

The vulnerability name is a short description that summarizes the nature of the problem and the affected software product. While the name may include a clause describing the impact of the vulnerability, most names are focused on the nature of the defect that caused the problem to occur.

Users Review

From reader reviews:

Teresa Brown:

Nowadays reading books become more than want or need but also be a life style. This reading behavior give you lot of advantages. Associate programs you got of course the knowledge your information inside the book this improve your knowledge and information. The info you get based on what kind of reserve you read, if you want attract knowledge just go with education and learning books but if you want feel happy read one with theme for entertaining including comic or novel. The actual The CERT C Secure Coding Standard is kind of book which is giving the reader unforeseen experience.

Jonathan Smith:

Hey guys, do you wishes to finds a new book to see? May be the book with the headline The CERT C Secure Coding Standard suitable to you? Typically the book was written by famous writer in this era. The particular book untitled The CERT C Secure Coding Standardis the main of several books this everyone read now. This book was inspired lots of people in the world. When you read this book you will enter the new shape that you ever know before. The author explained their strategy in the simple way, so all of people can easily to understand the core of this book. This book will give you a wide range of information about this world now. So you can see the represented of the world with this book.

Christine Emmons:

Playing with family in the park, coming to see the water world or hanging out with good friends is thing that usually you may have done when you have spare time, in that case why you don't try factor that really opposite from that. One particular activity that make you not experience tired but still relaxing, trilling like on roller coaster you already been ride on and with addition of knowledge. Even you love The CERT C Secure Coding Standard, you can enjoy both. It is good combination right, you still wish to miss it? What kind of hang type is it? Oh can occur its mind hangout men. What? Still don't buy it, oh come on its called reading friends.

Christie Rich:

This The CERT C Secure Coding Standard is brand-new way for you who has fascination to look for some information since it relief your hunger of information. Getting deeper you onto it getting knowledge more you know otherwise you who still having small amount of digest in reading this The CERT C Secure Coding Standard can be the light food for you personally because the information inside this kind of book is easy to get by simply anyone. These books build itself in the form which is reachable by anyone, yep I mean in the e-book type. People who think that in publication form make them feel sleepy even dizzy this reserve is the answer. So there is no in reading a book especially this one. You can find what you are looking for. It should be here for anyone. So , don't miss the idea! Just read this e-book kind for your better life along with knowledge.

Read The CERT C Secure Coding Standard By Robert C. Seacord for online ebook

The CERT C Secure Coding Standard By Robert C. Seacord Free PDF d0wnl0ad, audio books, books to read, good books to read, cheap books, good books, online books, books online, book reviews epub, read books online, books to read online, online library, greatbooks to read, PDF best books to read, top books to read The CERT C Secure Coding Standard By Robert C. Seacord books to read online.

Online The CERT C Secure Coding Standard By Robert C. Seacord ebook PDF download

The CERT C Secure Coding Standard By Robert C. Seacord Doc

The CERT C Secure Coding Standard By Robert C. Seacord Mobipocket

The CERT C Secure Coding Standard By Robert C. Seacord EPub

6PRGBY7ACMW: The CERT C Secure Coding Standard By Robert C. Seacord