JPCERT CC®

C/C++ セキュアコーディングセミナー 2010年度版

CERT C セキュアコーディングスタンダード

JPCERT コーディネーションセンター

JPCERT CC®

セキュアコーディングスタンダード概論

セキュアコーディングスタンダードって?
中身を見てみよう
今後の展開

セキュアコーディングスタンダード各論

内容をより詳しく見ていきます

セキュアコーディングスタンダード活用

どんな活用ができるか、そのアイディアを語ります

Copyright© 2011 JPCERT/CC All rights reserved.

今日お話する内容

JPCERT CC®

セキュアコーディングスタンダード概論

セキュアコーディングスタンダードって?
中身を見てみよう
今後の展開

セキュアコーディングスタンダード各論

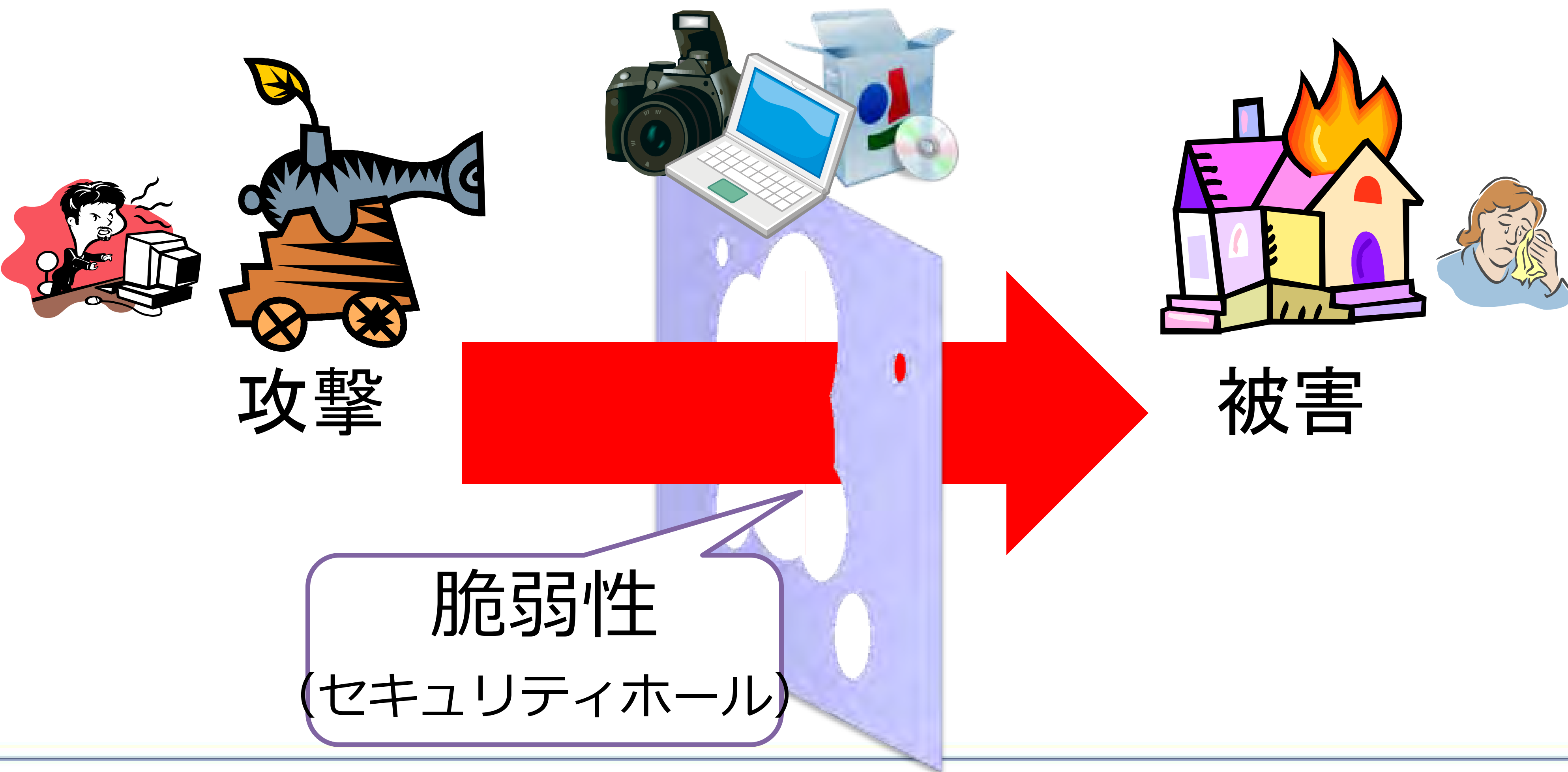
内容をより詳しく見ていきます

セキュアコーディングスタンダード活用

どんな活用ができるか、そのアイディアを語ります

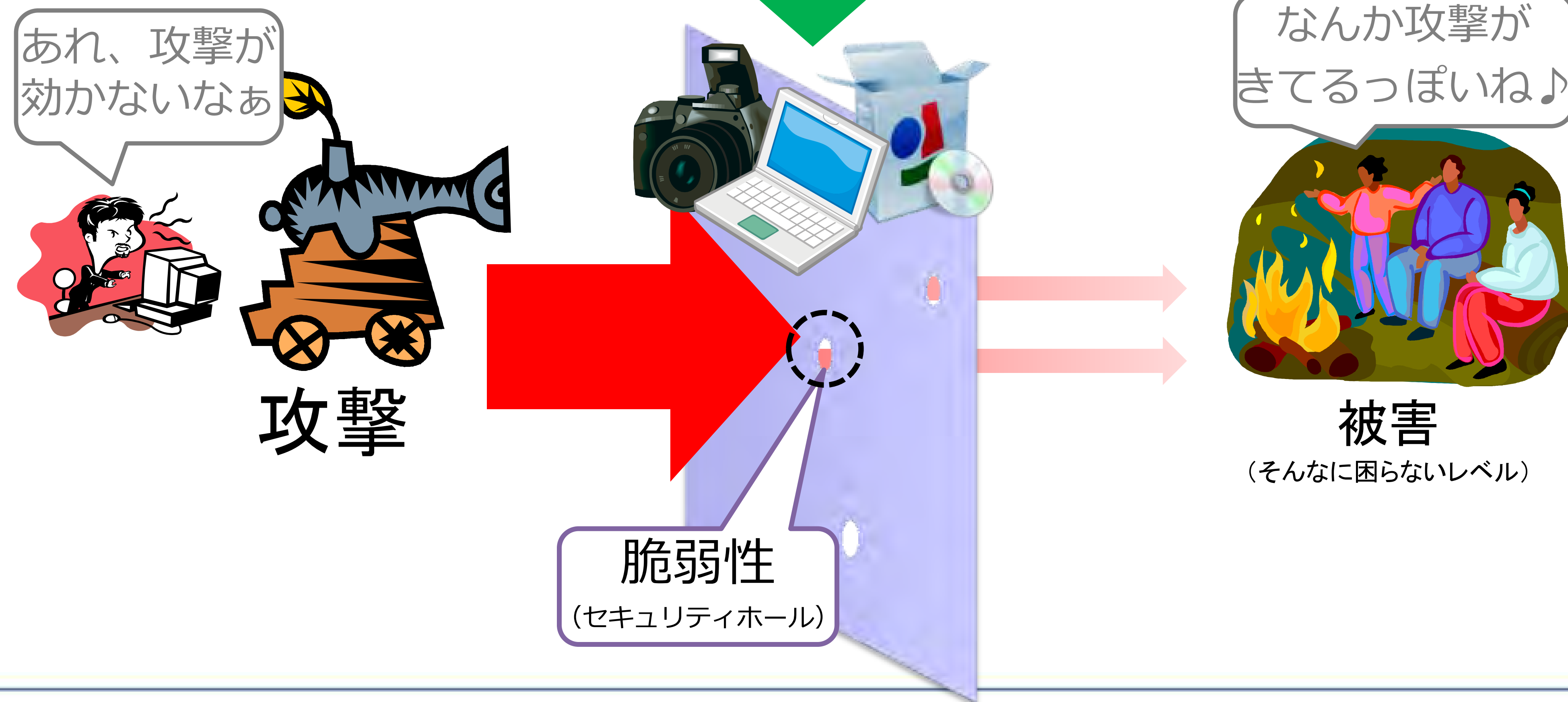
Copyright© 2011 JPCERT/CC All rights reserved.

Secure Coding Standard って?(1)
製品の脆弱性



Copyright© 2011 JPCERT/CC All rights reserved.

Secure Coding Standard って?(2)
脆弱性を減らし、被害を小さくする



Copyright© 2011 JPCERT/CC All rights reserved.

Secure Coding Standard って? (3)
プログラミング言語の問題

JPCERT CC®

C言語の精神:
プログラマを信頼し、やりたいことができるように

言語仕様に存在する未規定、未定義、実装定義事項

言語仕様の詳細を知らないプログラマが
未定義動作などの問題のあるコードを書き、
脆弱性を作り込む

セキュアなコードを書くための
道しるべ・ガイドラインが必要

Copyright© 2011 JPCERT/CC All rights reserved.

Secure Coding Standard って? (4)
ガイドラインって?

JPCERT CC®

Coding Style

Coding Standard

GNU coding standards
http://www.gnu.org/prep/standards/

Programming Style
http://en.wikipedia.org/wiki/Programming_style

Google-styleguide
http://code.google.com/p/google-styleguide/

FLEX SDK coding conventions and best practices
http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions

Zend Framework PHP 標準コーディング規約
http://framework.zend.com/manual/ja/coding-standard.html

C Style and Coding Standards for SunOS
http://opensolaris.org/os/community/on/cstyle.ms.pdf

The NetBSD source code style guide
http://cvsweb.netbsd.org/bsdweb.cgi/~checkout~/src/share/misc/style?rev=1.46&content-type=text/plain

Code Conventions for the Java Programming Language
http://java.sun.com/docs/codeconv/

JPL Institutional Coding Standard for the C Programming Language
http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_ext.pdf

MISRA-C:2004
Guidelines for the use of the C language in critical systems

などなど…

Copyright© 2011 JPCERT/CC All rights reserved.

「CERT C セキュアコーディングスタンダード」とは?

セキュアなソフトウェアをつくるためのコーディングガイドライン集

- 攻撃可能な脆弱性を作り込まない
- コードの保守性向上
  = プログラマが理解しやすい
  = 移植性向上

対象:
ソフトウェア開発やメンテナンスに携わる人々

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

「CERT C セキュアコーディングスタンダード」とは?

- コンパイラ作者向けでなく、Cプログラマ向け

- できるだけOSや実行環境に依存しない
  = 環境依存な方法が重要な場合、おもに
  POSIX(Unix)およびWindowsにおけるコード例を示す

- C99に基づく

- 今後の開発に役立つガイドラインが第一目的
  = C99環境における既存のレガシーコードへの対応が第二目的

Copyright© 2011 JPCERT/CC All rights reserved.

Secure Coding Standard って? (7)
GNU Coding Standards との比較

GNU Coding Standards との比較...

GNU Coding Standards

- GNUシステムをクリーンに、一貫性のある、インストールしやすいものにするため
  ("to make the GNU system clean, consistent, and easy to install")
- ポータブルで頑丈で高信頼性なプログラムを書くためのガイドにもなる
  ("also be read as a guide to writing portable, robust, and reliable programs")

CERT C セキュアコーディングスタンダード

- C を使う全ての人のために
- 攻撃に使われるような脆弱性を作り込まないようにするためのガイ
  ポータブル、頑丈、高信頼性と重なる部分あり

Copyright© 2011 JPCERT/CC All rights reserved.

MISRA-C:2004 との比較...

MISRA-C:2004

(Guidelines for the use of the C language in critical systems)

- 組み込みや重要インフラ系システムにおけるC使用に関するガイドライン
- 安全性、移植性、信頼性向上のため
- C90に基づく (C99対応は現在進行中)

CERT C セキュアコーディングスタンダード

- OSや実行環境などを特定せずC を使う全ての人のために
- 脆弱性をつくりこまないように → 安全性、移植性、信頼性と重なる
- C99に基づく

MISRA-C:2004
カテゴリ数: 21
ルール数: 141 (必要121, 推奨20)

CERT C セキュアコーディングスタンダード
カテゴリ数: 16
ルール数: 293 (必要114, 推奨179)

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

CERT/CC Secure Coding Initiative

言語仕様関係者、コンパイラベンダ、開発者など、
多様な人々を含むコミュニティベースでの議論と開発

CERT/CC のWikiサイト上で開発中

https://www.securecoding.cert.org/

JPCERT/CC にて日本語版
https://www.jpcert.or.jp/sc-rules/

実際に発見された脆弱性を反映
商用ソースコード解析ツールも対応

coverity, Fortify, Klocwork, LDRA, ...

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

CERT C Secure Coding Standard 日本語版
(https://www.jpcert.or.jp/sc-rules/)

- どんなセクションがあるか
- ルールとレコメンデーション
- 各ページの構成

(説明, 違反コード, 適合コード, リスク評価, 参考情報)

CERT C セキュアコーディングスタンダード
カテゴリ数: 16
ルール数: 293 (必要114, 推奨179)

Copyright© 2011 JPCERT/CC All rights reserved.

中身を見てみよう: どんなセクションがあるか

JPCERT CC®

01.プリプロセッサ(PRE)
02. 宣言と初期化(DCL)
03. 式(EXP)
04. 整数(INT)
05. 浮動小数点(FLP)
06. 配列(ARR)
07. 文字と文字列(STR)
08. メモリ管理(MEM)
09. 入出力(FIO)
10. 環境(ENV)
11. シグナル(SIG)
12. エラー処理(ERR)
13. Application Programming Interface(API)
49. 雑則(MSC)
50. POSIX(POS)

内容に応じて分類

14. 並行性(CON)

Copyright© 2011 JPCERT/CC All rights reserved.

中身を見てみよう: ルールとレコメンデーション

JPCERT CC®

☆ ルール

○ コーディング作法への違反が、攻撃可能な脆弱性を生み出すセキュリティ上の欠陥につながる可能性がある。

○ コーディング作法へ適合しているかどうかを、自動解析、形式的方法、手作業によるソースコード検査などを通じて確認することができる。

☆ レコメンデーション

○ コーディング作法を適用することで、システムのセキュリティが高まる可能性がある。

○ コーディング作法がルールと見なされるための要件を、一つ以上満たすことができない。

https://www.jpcert.or.jp/sc-rules/00.introduction.html

Copyright© 2011 JPCERT/CC All rights reserved.

中身を見てみよう: ルールとレコメンデーション

☆ ルール

セキュリティを確保するために必ず適用すべきもの

○ スタンダード準拠の必須項目

○ 適合していることを確認できる

☆ レコメンデーション

セキュリティを向上させるガイドライン、提案

○ スタンダード準拠には必須ではない

○ 必ずしも攻撃につながらない、適合の確認が難しい、など

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

識別番号とタイトル

カテゴリ名 番号-C　　例: INT01-C, EXP32-C

0～29までの番号はレコメンデーション
30以降の番号はルール

説明

違反コード

適合コード

(例外)

リスク評価

参考情報

Copyright© 2011 JPCERT/CC All rights reserved.

中身を見てみよう: 各ガイドラインの構成(2)

JPCERT CC®

識別番号とタイトル

説明

内容の説明
「～すべきである」「～すべきでない」

違反コード

簡単な違反コード例

適合コード

違反コードの修正例や別のアプローチのコード例など

(例外)

このガイドラインにしたがう必要のない状況についての説明

リスク評価
参考情報

Copyright© 2011 JPCERT/CC All rights reserved.

中身を見てみよう: 各ガイドラインの構成(3)

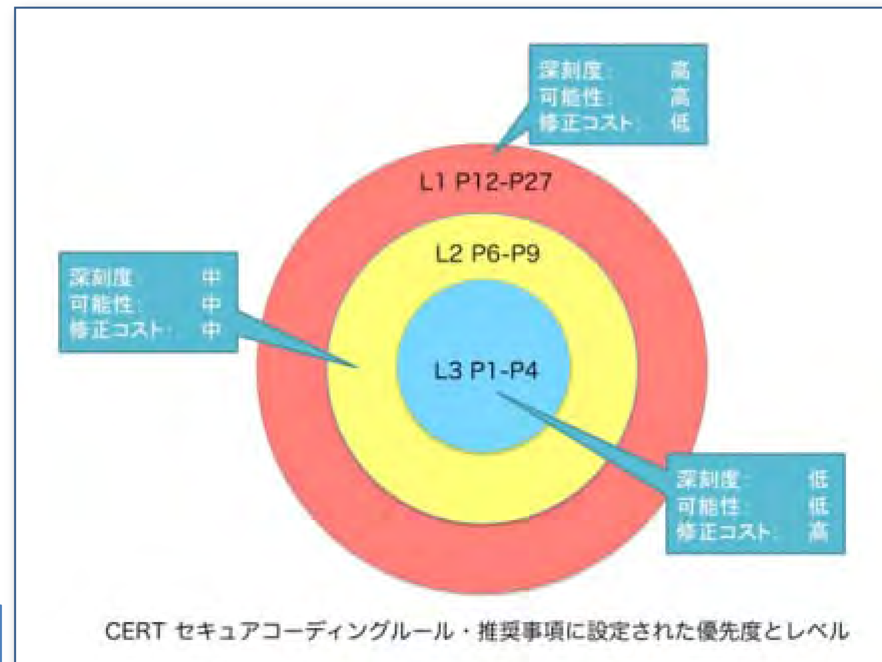
識別番号とタイトル
説明
違反コード
適合コード
例外
リスク評価

深刻度、攻撃可能性、修正コストの観点から評価

参考情報

C99の関連セクション、解説記事、関連するCWE番号など

この評価値により、準拠すべきガイドラインに優先順位をつけられる



CERT セキュアコーディングルール・推奨事項に設定された優先度とレベル

https://www.jpcert.or.jp/sc-rules/00.introduction.html

Copyright© 2011 JPCERT/CC All rights reserved.

例: INT35-C: (1/6)
整数式をより大きなサイズの整数に対して比較や代入をする際には、事前に演算後のサイズで評価する

JPCERT CC®

整数式の結果の値を、それよりサイズの大きな整数型と比較したりあるいは代入するときは、まず、式中のオペランドのいずれかを大きな整数型にキャストし、 その型で整数式が評価されるようにしなくてはならない。

違反コード

以下のコードは、size_t が 符号無し32ビットの値をとり、long long が64ビットの値をとるシステムではルールに違反する。この例でプログラマは、SIZE_MAX と length + BLOCK_HEADER_SIZE を比較することでラップアラウンドが発生するかどうかをテストしている。length は size_t として宣言されているため、加算は32ビット演算として実行され、その結果整数オーバーフローが発生する可能性がある。SIZE_MAX との比較は常に偽と判定される。ラップアラウンドが発生すると、malloc() は mBlock に十分なメモリ領域を割り当てず、その結果バッファオーバーフローにつながる恐れがある。

```
enum { BLOCK_HEADER_SIZE = 16 };
void *AllocateBlock(size_t length) {
  struct memBlock *mBlock;
  if (length + BLOCK_HEADER_SIZE > (unsigned long long)SIZE_MAX)
      return NULL;
  mBlock = (struct memBlock *)malloc( length + BLOCK_HEADER_SIZE );
  if (!mBlock) return NULL;
  /* ブロックヘッダをデータで埋めて、データを返す */
  return mBlock;
}
```

この条件を検出するコンパイラもある。

Copyright© 2011 JPCERT/CC All rights reserved.

例: INT35-C: (2/6)
整数式をより大きなサイズの整数に対して比較や代入をする際には、
事前に演算後のサイズで評価する

JPCERT CC®

適合コード(アップキャスト)

以下の解決法では、length オペランドは unsigned long long にキャストされ、このサイズで加算が行われるようにしている。

```
enum { BLOCK_HEADER_SIZE = 16 };
void *AllocateBlock(size_t length) {
  struct memBlock *mBlock;
  if ((unsigned long long)length + BLOCK_HEADER_SIZE >
SIZE_MAX) {
    return NULL;
  }
  mBlock = (struct memBlock *)malloc(
      length + BLOCK_HEADER_SIZE );
  if (!mBlock) return NULL;
  /* ブロックヘッダをデータで埋めて、データを返す */
  return mBlock;
}
```

このようなラップアラウンドのテストが有効なのは、sizeof(unsigned long long) > sizeof(size_t) の場合に限る。仮に size_t と unsigned long long 型が64ビットの符号無し型として表現されているとすると、加算演算の結果は unsigned long long 型の値として表現できない可能性がある。

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

適合コード(式の再配置)

以下のコードでは、length は SIZE_MAX から減算されており、ラップアラウンドが発生しないことを保証している。「INT30-C. 符号無し整数の演算結果がラップアラウンドしないようにする」を参照のこと。

```
enum { BLOCK_HEADER_SIZE = 16 };
void *AllocateBlock(size_t length) {
  struct memBlock *mBlock;
  if (SIZE_MAX - length < BLOCK_HEADER_SIZE) {
    return NULL;
  }
  mBlock = (struct memBlock *)malloc(
       length + BLOCK_HEADER_SIZE );
  if (!mBlock) return NULL;
  /* ブロックヘッダをデータで埋めて、データを返す */
  return mBlock;
}
```

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

違反コード

以下のコードでは、プログラマは unsigned long long 型変数 alloc に cBlocks * 16 の結果を代入することでラッピングを回避しようとしている。

```
void* AllocBlocks(size_t cBlocks) {
  if (cBlocks == 0) return NULL;
  unsigned long long alloc = cBlocks * 16;
  return (alloc < UINT_MAX) ? malloc(cBlocks * 16) : NULL;
}
```

このコードには2つの問題がある。1つ目の問題は、unsigned long long が size_t より少なくとも4ビット大きい実装を想定しているということである。2つ目の問題は、size_t が32ビット値であり unsigned long long が64ビット値である処理系を想定すると、この演算が C99 に準拠するためには、2つの32ビット値の乗算結果が32ビットにならなくてはならないということである。この乗算が原因で発生する整数オーバーフローは すべて検出されず、式 alloc < UINT_MAX は常に真になる。

Copyright© 2011 JPCERT/CC All rights reserved.

例: INT35-C: (5/6)

整数式をより大きなサイズの整数に対して比較や代入をする際には、
事前に演算後のサイズで評価する

JPCERT CC®

適合コード

以下のコードでは、cBlocks オペランドは unsigned long long にキャストされ、このサイズで乗算が行われるようにしている。

```
static_assert(
  CHAR_BIT * sizeof(unsigned long long) >=
  CHAR_BIT * sizeof(size_t) + 4,
  "乗算後にラップアラウンドを検知することはできない"
);

void* AllocBlocks(size_t cBlocks) {
  if (cBlocks == 0) return NULL;
  unsigned long long alloc = (unsigned long long)cBlocks *
16;
  return (alloc < UINT_MAX) ? malloc(cBlocks * 16) : NULL;
}
```

ただし、このコードは unsigned long long 型が size_t よりも少なくとも4ビット大きくなければオーバーフローを回避することはできない。

Copyright© 2011 JPCERT/CC All rights reserved.

例: INT35-C: (6/6)
整数式をより大きなサイズの整数に対して比較や代入をする際には、事前に演算後のサイズで評価する

JPCERT CC®

リスク評価

整数値をより大きなサイズの整数型の値と比較したり代入する前に型変換しなかった場合、攻撃者が脆弱なプロセスの実行権限で任意のコードを実行可能なソフトウェア脆弱性を引き起こす可能性がある。

ルール	深刻度	可能性	修正コスト	優先度	レベル
INT35-C	高	高	中	P18	L1

自動検出

CERT C Rule Packを適用したForfity SCAバージョン5.0はこのルールの違反を検出することができる。

Compass/ROSE はこのルールの違反を検出することができる。次のような条件を満たすパターン（a op1 b) op2 c を探す。

- c の型のサイズは a や b のサイズよりも大きい
- a と b のどちらも c の型とは異なる
- op2 が代入もしくは比較演算子である

参考情報

[Dowd 06] Chapter 6, "C Language Issues"

[ISO/IEC 9899:1999] Section 6.3.1, "Arithmetic Operands"

[ISO/IEC PDTR 24772] "FLC Numeric Conversion Errors"

[MITRE 07] CWE ID 681, "Incorrect Conversion between Numeric Types," and CWE ID 190, "Integer Overflow (Wrap or Wraparound)"

[Seacord 05a] Chapter 5, "Integer Security"

翻訳元

INT35-C. Evaluate integer expressions in a larger size before comparing or assigning to that size

Copyright© 2011 JPCERT/CC All rights reserved.

おすすめリーディングマップ

次の一歩!

JPCERT CC®

04. 整数(INT)
07. 文字と文字列(STR)

基本データ型について
しっかりおさえよう!

06. 配列(ARR)
08. メモリ管理(MEM)

重要なデータ構造として、
配列と動的メモリ管理を
チェック!

01. プリプロセッサ(PRE)
02. 宣言と初期化(DCL)
03. 式(EXP)
10. 環境(ENV)
09. 入出力(FIO)

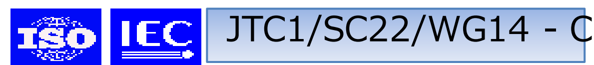
05. 浮動小数点(FLP)
11. シグナル(SIG)
12. エラー処理(ERR)
13. (API)
50. POSIX(POS)

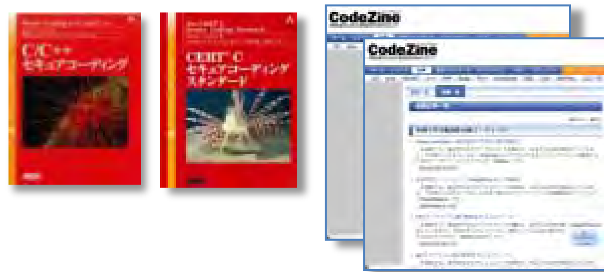
Copyright© 2011 JPCERT/CC All rights reserved.

今後の展開

JPCERT CC

- ソースコード解析ツールの対応
- 標準化活動
- 他の言語への展開
- セミナーによる普及啓発
- メディアへの展開





Copyright© 2011 JPCERT/CC All rights reserved.

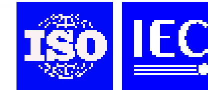
ソースコード解析ツールの対応

- Coverity (http://www.coverity.com/)
- Fortify (http://www.fortify.com/)
- Klocworks (http://www.klocwork.com/solutions/security-coding-standards/)
- LDRA (http://www.ldra.com/jp/certc.asp)
- ......
- CERT ROSE Checkers (http://sourceforge.net/projects/rosecheckers/)

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

標準化活動



JTC1/SC22/WG14 - C

C Secure Coding Guidelines Study Group

ツールで解析可能なコーディングガイドラインについて議論

CERT/CC の wiki サイトを使って活動中

参考: C Secure Coding Guidelines Study Group WG14 Liason Report (2010-03-16)

(http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1450.pdf)

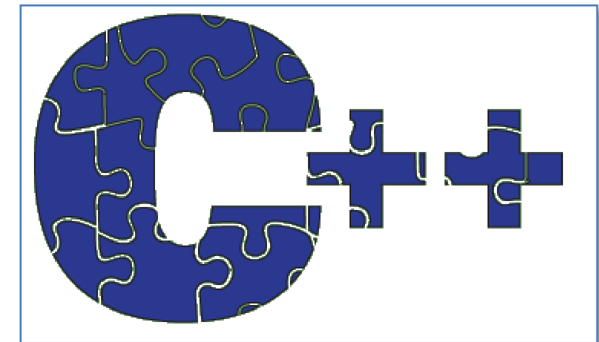
- C1x が確定するまでは、WG14のリソースを削らないよう、study group として議論する形で続ける
- 成果は type 2 technical report としてまとめる予定
- C1x 以降の言語仕様にとりこむかどうかはあらためて検討

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

他の言語への展開

C++ - The CERT C++ Secure Coding Standard
(https://www.securecoding.cert.org/confluence/display/cplusplus)



Java - The CERT Oracle Secure Coding Standard for Java
(https://www.securecoding.cert.org/confluence/display/java)

Copyright© 2011 JPCERT/CC All rights reserved.

今後の展開(4) - セミナーによる普及啓発

JPCERT CC®

一般向けオープンセミナ：

2010年度は福岡、大阪、名古屋、東京、札幌で開催

企業・組織向け出張セミナ(有料)：

これまで、国内大手家電メーカ、機器メーカ、ソフトウェア会社等で多数実施しています。

Menu：

1. セキュアコーディング概論
2. 文字列
3. 整数
4. 動的メモリ管理
5. ファイルI/O
6. 書式指定文字列
7. **Secure Coding Standards**
8. 静的解析ツール

Copyright© 2011 JPCERT/CC All rights reserved.

今後の展開(5) - メディアへの展開

Web、書籍など各種メディアを通じた啓発活動

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

セキュアコーディングスタンダード概論

セキュアコーディングスタンダードって?
中身を見てみよう
今後の展開

セキュアコーディングスタンダード各論

内容をより詳しく見ていきます

セキュアコーディングスタンダード活用

どんな活用ができるか、そのアイディアを語ります

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

セキュアコーディングスタンダード日本語版
ルールおよびレコメンデーション一覧

(2011年03月31日現在)

01.プリプロセッサ(PRE)
02. 宣言と初期化(DCL)
03. 式(EXP)
04. 整数(INT)
05. 浮動小数点(FLP)
06. 配列(ARR)
07. 文字と文字列(STR)
08. メモリ管理(MEM)
09. 入出力(FIO)
10. 環境(ENV)
11. シグナル(SIG)
12. エラー処理(ERR)
13. Application Programming Interface(API)
14. 並行性(CON)
49. 雑則(MSC)
50. POSIX(POS)

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

プリプロセッサの機能に関連するガイドライン

- マクロ定義において注意すべきこと
- マクロ呼び出しで注意すべきこと
- 3文字表記の使い方に関する注意
- ヘッダファイルに関する注意

など

Copyright© 2011 JPCERT/CC All rights reserved.

01.プリプロセッサ(PRE)

JPCERT CC®

レコメンデーション

PRE00-C. 関数形式マクロよりもインライン関数やスタティック関数を使う
PRE01-C. マクロ内の引数名は括弧で囲む
PRE02-C. マクロ置換リストは括弧で囲む
PRE03-C. 型をエンコードするには define よりも typedef を選ぶ
PRE04-C. 標準ヘッダファイル名を再利用しない
PRE05-C. 字句の結合や文字列化を行う際にはマクロ置換をよくよく理解して行う
PRE06-C. ヘッダファイルはインクルードガードで囲む
PRE07-C. "??" の繰り返しは避ける
PRE08-C. ヘッダファイル名が一意であることを保証する
PRE09-C. セキュアな関数をよりセキュアでない関数に置き換えない
PRE10-C. 複数にわたる文からなるマクロは do-while ループで包む
PRE11-C. 単一文からなるマクロ定義をセミコロンで終端しない
PRE12-C. 数値定数は移植性のある方法で定義する

ルール

PRE30-C. 文字列連結によってユニバーサル文字名を作成しない
PRE31-C. 安全でないマクロの引数では副作用を避ける

Copyright© 2011 JPCERT/CC All rights reserved.

例: PRE31-C: 安全でないマクロの引数では副作用を避ける(1/2)

JPCERT CC®

安全でない関数型マクロとは、コードに展開される際に引数を2回以上評価するか、あるいはまったく評価しないもののこと。
代入、インクリメント、デクリメント、volatileアクセス、入出力、その他の副作用(副作用を引き起こす可能性のある関数呼び出しを含む)を持つ引数を使って、安全でないマクロを呼び出さないこと。

違反コード

```
#define ABS(x) (((x) < 0) ? -(x) : (x))
/* ...... */
m = ABS(++n); /* 未定義の動作 */
```

この ABS マクロは以下のように展開される。

```
m = (((++n)<0) ? -(++n) : (++n)); /* 未定義の動作 */
```

展開されたコードは、「EXP30-C. 副作用完了点間の評価順序に依存しない」に違反しており、未定義の動作となる。

Copyright© 2011 JPCERT/CC All rights reserved.

例: PRE31-C: 安全でないマクロの引数では副作用を避ける(2/2)

JPCERT CC®

適合コード1

副作用を持った引数を使わないコードにする。

```
#define ABS(x) (((x) < 0) ? -(x) : (x)) /* 安全でないマクロ定義 */
/* ...... */
++n;
m = ABS(n);
```

適合コード2

inline 関数にする。

```
inline int abs(int x) {
  return (((x) < 0) ? -(x) : (x));
}
/* ...... */
m = abs(++n);
```

Copyright© 2011 JPCERT/CC All rights reserved.

変数や関数などの宣言と初期化に関する
ガイドライン

- 保守しやすい名前付けをする
- 型修飾子(const、restrict、volatile)に関する注意
- 有効範囲や記憶域期間は最小限に
- 可変引数関数に関する注意

など

Copyright© 2011 JPCERT/CC All rights reserved.

02. 宣言と初期化(DCL)

JPCERT CC®

レコメンデーション

DCL00-C. イミュータブルなオブジェクトは const 修飾する
DCL01-C. サブスコープで変数名を再利用しない
DCL02-C. 視覚的に区別できる識別子を使う
DCL03-C. 定数式の値をテストするには静的アサートを使う
DCL04-C. ひとつの宣言で2つ以上の変数を宣言しない
DCL05-C. typedef を使いコードの可読性を改善する
DCL06-C. プログラムロジックでリテラル値を表現するには意味のあるシンボル定数を使う
DCL07-C. 関数宣言子には適切な型情報を含める
DCL08-C. 定数定義間の関係は適切にコード化する
DCL09-C. errno エラーコードを返す関数は返り値をerrno_t型として定義する
DCL10-C. 可変引数関数の作成者と利用者の間の取り決めを維持する
DCL11-C. 可変引数関数に関連する型問題について理解する
DCL12-C. 抽象データ型は opaque な型を使って実装する
DCL13-C. 関数の引数が関数自身によって変更されない値を参照するポインタならば、関数の引数をconstとして宣言する
DCL15-C. 外部結合を必要としないファイル有効範囲のオブジェクトや関数は static 宣言する
DCL16-C. long 値を表すには小文字の 'l' ではなく大文字の 'L' を使う
DCL17-C. volatile 修飾された変数が間違ってコンパイルされることに注意
DCL18-C. 0で始まる整数定数に注意
DCL19-C. 全ての変数および関数に対して最小限の有効範囲を用いる

ルール

DCL30-C. 適切な記憶域期間を持つオブジェクトを宣言する
DCL31-C. 識別子は宣言してから使用する
DCL32-C. 相互に可視である識別子が一意であることを保証する
DCL33-C. 関数引数中で restrict 修飾されたコピー元およびコピー先が、重複したオブジェクトを参照しないようにする
DCL34-C. キャッシュできないデータには volatile を使う
DCL35-C. 関数定義と一致しない型で関数を呼び出さない
DCL36-C. 矛盾する結合の種類を使用して識別子を宣言しない

Copyright© 2011 JPCERT/CC All rights reserved.

例: DCL03-C: 定数式の値をテストするには静的アサートを使う(1/3)

assert()マクロは実行時診断を行う。また、引数式が偽の場合、abort()関数を呼び出す。したがって、動作し続けることが要求されるサーバプログラムや組み込みシステムには向いていない。

違反コード

```
struct timer {
  uint8_t MODE;
  uint32_t DATA;
  uint32_t COUNT;
};

int func(void){
  assert(offsetof(timer, DATA) == 4);
}
```

struct timer の定義と assert() 呼び出しの関連がコード上分かりづらい。
assert()による診断はコンパイル時には行われない。実行時に行われる。
assert()呼び出しを含む部分が実行されてはじめて診断が行われる。

Copyright© 2011 JPCERT/CC All rights reserved.

例: DCL03-C: 定数式の値をテストするには静的アサートを使う(2/3)

適合コード

```
#define JOIN(x, y) JOIN_AGAIN(x, y)
#define JOIN_AGAIN(x, y)   x ## y

#define static_assert(e) ¥
typedef char JOIN(assertion_failed_at_line_,
__LINE__)[(e)?1:-1]

struct timer {
  uint8_t MODE;
  uint32_t DATA;
  uint32_t COUNT;
};

static_assert(offsetof(timer, DATA) == 4);
```

struct timer の定義と static_assert() を一緒に置いておける。
static_assert()による診断はコンパイル時に行われる。

Copyright© 2011 JPCERT/CC All rights reserved.

例: DCL03-C: 定数式の値をテストするには静的アサートを使う(3/3)

C1xでは、コンパイル時診断を実現する機能として `_Static_assert()` が導入される。

参考: n1494の "6.7.10 Static assertions" および "7.2 Diagnostics <assert.h>"

```
_Static_assert(constant-expression, string-literal);
```

`static_assert()` は `_Static_assert()` に展開されるマクロとして `<assert.h>` で定義される。

C1xの2010-06-25ドラフト
http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1494.pdf

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

C言語の式(expression)の扱いに関するガイドライン

- 副作用に注意する
- ポインタ演算を正しく使う
- 構造体の扱い
- 型情報、型修飾子の扱い

など

Copyright© 2011 JPCERT/CC All rights reserved.

レコメンデーション

EXP00-C. 括弧を使用して演算の優先順位を指定する
EXP01-C. ポインタが参照する型のサイズを求めるのにポインタのサイズを使わない
EXP02-C. 論理 AND 演算子および論理 OR 演算子のショートサーキット動作について注意する
EXP03-C. 構造体のサイズが構造体のメンバのサイズの和に等しいと決めてかからない
EXP04-C. 構造体を含むバイト単位の比較を行わない
EXP05-C. const 修飾をキャストではずさない
EXP06-C. sizeof 演算子のオペランドは副作用を持たせない
EXP07-C. 式中で、定数の値を仮定して定数を使うメリットを損なわない
EXP08-C. ポインタ演算は正しく使用する
EXP09-C. 型や変数のサイズは sizeof を使って求める
EXP10-C. 部分式の評価順序や副作用の発生順序に依存しない
EXP11-C. ある型を期待している演算子に互換性のない型のデータを渡さない
EXP12-C. 関数の返り値を無視しない
EXP13-C. 関係演算子および等価演算子は、結合則が成り立たないものとして扱う
EXP14-C. char 型や short 型の値に対してビット単位の演算を行う際には整数拡張(integer promotion)が行われることに注意

ルール

EXP30-C. 副作用完了点間の評価の順序に依存しない
EXP31-C. assert() のなかでは副作用を避ける
EXP32-C. volatile 修飾子をキャストにより無効にしない
EXP33-C. 未初期化のメモリを参照しない
EXP34-C. NULL ポインタを参照しない
EXP35-C. 関数呼び出しの結果に含まれる配列に、次の副作用完了点より後でアクセスしたり変更を行わない
EXP36-C. ポインタをより厳密にアラインされるポインタ型に変換しない
EXP37-C. API が意図した引数で関数を呼び出す
EXP38-C. ビットフィールドメンバや無効な型で offsetof() を呼び出さない
EXP39-C. 適合しない型のポインタを使って変数にアクセスしない

Copyright© 2011 JPCERT/CC All rights reserved.

例: EXP35-C:
関数呼び出しの結果に含まれる配列に、次の副作用完了点より後でアクセスしたり変更を行わない(1/2)

JPCERT CC®

違反コード

```
#include <stdio.h>
struct X { char a[6]; };

struct X addressee(void) {
  struct X result = { "world" };
  return result;
}

int main(void) {
  printf("Hello, %s!¥n",
addressee().a);
  return 0;
}
```

- 関数の返り値の生存期間は次の副作用完了点まで。
- 関数の引数は値渡し
- 関数の引数としての配列は実際にはポインタが(コピーされて)渡される

printf()呼び出しの時点では addressee() の返り値は無効
addressee().a が指しているアドレスは addressee() 内部でのみ有効なものかも
C++ では問題ないコード

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

適合コード

```
#include <stdio.h>
struct X { char a[6]; };

struct X addressee(void) {
  struct X result = { "world" };
  return result;
}

int main(void) {
  struct X my_x = addressee();
  printf("Hello, %s!¥n", my_x.a);
  return 0;
}
```

- addressee() の返り値を my_x に代入
- struct X のデータがコピーされる
- my_x.a の値は printf()呼び出し時に有効!

C1x では、C++ に合わせて生存期間に関する改訂が行われる予定.
(6.2.4 の paragraph8)

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

整数型データの扱いに関するガイドライン

- 整数変換のルール
- 整数オーバフローについて
- シフト演算に関する注意
- ポインタとの変換
- 文字列データからの変換にはstrtol系を

など

Copyright© 2011 JPCERT/CC All rights reserved.

04. 整数(INT)

JPCERT CC®

レコメンデーション

INT00-C. 処理系のデータモデルについて理解する
INT01-C. オブジェクトのサイズを表現するすべての整数値に rsize_t もしくは size_t を使用する
INT02-C. 整数変換のルールを理解する
INT03-C. セキュアな整数ライブラリを使用する
INT04-C. 信頼できない入力源から取得した整数値は制限する
INT05-C. 可能性のあるすべての入力を処理できない入力関数を使って文字データを変換しない
INT06-C. 文字列トークンを整数に変換するには strtol() 系の関数を使う
INT07-C. 数値には符号の有無を明示した文字型のみを使用する
INT08-C. 全ての整数値が範囲内にあることを確認する
INT09-C. 列挙定数が一意の値に対応することを保証する
INT10-C. % 演算子を使用する際、結果の剰余が正であると想定しない
INT11-C. ポインタ型から整数型への変換やその逆の変換は注意して行う
INT12-C. 式中で使用される単なるintのビットフィールドの型について想定しない
INT13-C. ビット単位の演算子は符号無しオペランドに対してのみ使用する
INT14-C. 同じデータに対してビット単位の演算と算術演算を行わない
INT15-C. プログラマ定義の整数型に対する書式付き入出力には、intmax_t もしくは uintmax_t を使用する
INT16-C. 符号付き整数の表現形式を想定しない
INT17-C. 処理系に依存しない方法で整数定数を定義する

ルール

INT30-C. 符号無し整数の演算結果がラップアラウンドしないようにする
INT31-C. 整数変換によってデータの消失や解釈間違いが発生しないことを保証する
INT32-C. 符号付き整数演算がオーバーフローを引き起こさないことを保証する
INT33-C. 除算および剰余演算がゼロ除算エラーを引き起こさないことを保証する
INT34-C. 負のビット数、あるいはオペランドのビット数以上シフトしない
INT35-C. 整数式をより大きなサイズの整数に対して比較や代入をする際には、事前に演算後のサイズで評価する

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

浮動小数点データの扱いに関するガイドライン

- 浮動小数点数の精度の限界に注意
- 浮動小数点エラー
- 数学関数のエラー処理

など

Copyright© 2011 JPCERT/CC All rights reserved.

05. 浮動小数点(FLP)

JPCERT CC®

レコメンデーション

FLP00-C.浮動小数点数の限界について理解していること
FLP01-C.浮動小数点型の式に対する再構成には注意すること
FLP02-C.厳密な計算が必要な場面で浮動小数点数の使用は避けること
FLP03-C.浮動小数エラーは検出し対処すること
FLP04-C.浮動小数点入力が例外値でないか検査する
FLP05-C.非正規化数を使用しない

ルール

FLP30-C.浮動小数点変数をループカウンタに使用しない
FLP31-C.実数値を引数にとる関数を複素数値で呼び出さない
FLP32-C.数学関数における定義域エラーおよび値域エラーを防止または検出する
FLP33-C.浮動小数点数の演算時には整数を浮動小数点数に変換する
FLP34-C.浮動小数点の型変換が新しい型の範 囲に収まるようにする
FLP35-C.浮動小数点数値を比較する際には精度を考慮する

Copyright© 2011 JPCERT/CC All rights reserved.

例: FLP03-C: 浮動小数エラーは検出し対処すること(1/5)

JPCERT CC®

浮動小数点型の演算では、エラーが発生してもプログラムが停止するとは限らない。
整数演算でのエラーはプログラム停止となることが多い。

```
int j = 0;
int iResult = 1 / j;   /* 0除算は未定義動作 */
```

しかし、浮動小数点演算でのエラーでは停止しないことが多い。

```
double x = 0;
int dResult = 1 / x;   /* 0除算は未定義動作 */
```

C99では浮動小数点演算のエラー判別方法として、
fenv.h で規定される浮動小数点例外の機能を規定している。

Copyright© 2011 JPCERT/CC All rights reserved.

例: FLP03-C: 浮動小数エラーは検出し対処すること(2/5)

JPCERT CC®

違反コード

```
void fpOper_noErrorchecking(void){
  double a = 1e-40, b, c = 0.1;
  float x = 0, y;
  y = a;                         /* inexact and underflows */
  b = y / x;                     /* divide by zero operation */
  c = sin(30) * a;    /* inexact (loss of precision) */
}
```

Copyright© 2011 JPCERT/CC All rights reserved.

例: FLP03-C: 浮動小数エラーは検出し対処すること(3/5)

適合コード(C99)

```
#include <fenv.h>
#pragma   STDC   FENV_ACCESS   ON

void fpOper_fenv(void){
  double a = 1e-40, b, c = 0.1;
  float x = 0, y;
  int fpeRaised;

  feclearexcept(FE_ALL_EXCEPT);
  y = a;                        /* inexact
underflows */
  fpeRaised = fetestexcept(FE_ALL_EXCEPT);
  /* fpeRaised の値に応じてエラー処理 */
  ......
}
```

feclearexcept()で例外フラグをクリアする

fetestexcept()で例外発生をチェックする

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

適合コード(Windows)

_clearfp()で状態ワードの取得とクリアを行う

```
void fpOper_usingStatus(void){
  double a = 1e-40, b, c = 0.1;
  float x = 0, y;
  unsigned int rv = _clearfp();

  y = a;                          /* inexact and underflows */
  rv = _clearfp();
  /* rv の値に応じてエラー処理 */
  ......
}
```

Copyright© 2011 JPCERT/CC All rights reserved.

例: FLP03-C: 浮動小数エラーは検出し対処すること(5/5)

適合コード(Windows SEH)

```
void fpOper_usingSEH(void){
  double a = 1e-40, b, c = 0.1;
  float x = 0, y;
  unmask_fpsr();  /* _controlfp_s()による制御ワード初期化 */
  _try {
      y = a;                        /* inexact and underflows */
      ......
  }
  _except (_fpieee_flt(
                         GetExceptionCode(),
GetExceptionInformation(),
                         fpieee_handler)) {
        ...... fpieee_handler()がEXCEPTION_EXECUTE_HANDLERを
        ...... 返した場合の処理
  }
}
```

fpieee_handler() は
ユーザ定義のエラーハンドラ

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

配列データの扱いに関するガイドライン

- 配列の仕組みを理解する
- 配列のサイズを正しく求める
- 配列とポインタの関係
- 配列外アクセスに注意

など

Copyright© 2011 JPCERT/CC All rights reserved.

06. 配列(ARR)

JPCERT CC®

レコメンデーション

ARR00-C. 配列の仕組みを理解する
ARR01-C. 配列のサイズを求める際に sizeof 演算子をポインタに適用しない
ARR02-C. 初期化子が暗黙的にサイズを定義する場合であっても、配列のサイズは明示的に指定する

ルール

ARR30-C. 配列のインデックスが適切な範囲内にあることを保証する
ARR31-C. すべてのソースファイルにわたり一貫性のある配列表記を用いる
ARR32-C. 可変長配列のサイズ引数は適切な範囲内にあることを保証する
ARR33-C. コピーは必ず十分なサイズの記憶領域に対して行われることを保証する
ARR34-C. 式中の配列の型は互換性があることを保証する
ARR35-C. ループが配列の最後を越えて反復処理しない
ARR36-C. 異なる配列を指す2つのポインタに対して減算や比較を行わない
ARR37-C. 配列以外のオブジェクトを指すポインタに対して整数の加算や減算を行わない
ARR38-C. 結果の値が有効な配列要素を指さないような、ポインタに対する整数の加算や減算を行わない

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

文字や文字列データの扱いに関するガイドライン

- null 終端文字列の扱い
- 文字列リテラルの扱い
- 文字列とポインタの関係
- ワイド文字の文字列サイズ

など

Copyright© 2011 JPCERT/CC All rights reserved.

07. 文字と文字列(STR)

JPCERT CC®

レコメンデーション

STR00-C. 文字の表現には適切な型を使用する
STR01-C. 文字列の管理は一貫した方法で行う
STR02-C. 複雑なサブシステムに渡すデータは無害化する
STR03-C. null終端バイト文字列を不注意に切り捨てない
STR04-C. 基本文字集合にある文字を表すには単なる char を使用する
STR05-C. 文字列リテラルの参照には const へのポインタを使用する
STR06-C. strtok() が分割対象文字列を変更しないと想定しない
STR07-C. TR 24731 を使用し、文字列操作を行う既存のコードの脅威を緩和する
STR08-C. 新たに開発する文字列処理するコー ドには managed string を使用する

ルール

STR30-C. 文字列リテラルを変更しない
STR31-C. 文字データとnull終端文字を格納するために十分な領域を確保する
STR32-C. 文字列はnull終端させる
STR33-C. ワイド文字の文字列サイズは正しく求める
STR34-C. 文字データをより大きなサイズの整数型に変換するときは事前に unsigned 型に変換する
STR35-C. 長さに制限のないコピー元から固定長配列へデータをコピーしない
STR36-C. 文字列リテラルで初期化された文字配列の範囲を指定しない
STR37-C. 文字処理関数への引数は unsigned char として表現できなければならない

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

動的メモリ管理に関するガイドライン

- 必要なメモリサイズの計算に注意
- メモリの動的割り当てと解放に関する注意
- サイズ0のメモリ割り当ては避ける

など

Copyright© 2011 JPCERT/CC All rights reserved.

08. メモリ管理(MEM)

JPCERT CC®

レコメンデーション

MEM00-C. メモリの割り当てと解放は、同じ翻訳単位内の同一抽象レベルで行う
MEM01-C. free() の直後に新しい値をポインタに代入する
MEM02-C. メモリ割り当て関数の結果は、割り当てた型へのポインタに即座にキャストする
MEM03-C. 再利用可能なリソースに格納された機密情報は消去する
MEM04-C. サイズ 0 のメモリ割り当てを行わない
MEM05-C. 大きなスタックを割り当てない
MEM06-C. 機密情報はディスクに書き出さない
MEM07-C. calloc() の二つの引数を乗算した結果は size_t 型で表現できること
MEM08-C. realloc() は動的に割り当てられた配列のサイズ変更にしか使用してはならない
MEM09-C. メモリ割り当て関数がメモリを初期化すると想定しない
MEM10-C. ポインタ検証関数を定義して使用する
MEM11-C. ヒープ領域が無限にあると想定しない
MEM12-C. リソースの使用および解放の最中に発生するエラーで関数を終了する場合、goto 連鎖の使用を検討する

ルール

MEM30-C. 解放済みメモリにアクセスしない
MEM31-C. 動的に割り当てられたメモリは一度だけ解放する
MEM32-C. メモリ割り当てエラーを検出し、対処する
MEM33-C. フレキシブル配列メンバには正しい構文を使用する
MEM34-C. 動的に割り当てられたメモリのみを解放する
MEM35-C. オブジェクトに対して十分なメモリを割り当てる

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

(ファイルを含む)入出力に関するガイドライン

- 書式指定文字列の使用上の注意
- ファイル名の正規化
- ファイルのパーミションについて
- `fopen()`の使用上の注意
- ファイル名を引数にとる操作関数に注意

など

Copyright© 2011 JPCERT/CC All rights reserved.

09. 入出力(FIO)

JPCERT CC®

レコメンデーション

FIO00-C. 書式文字列は注意して作成する
FIO01-C. ファイル名を使用してファイルを識別する関数の使用に注意する
FIO02-C. 信頼できない情報源から取得したファイル名は正規化する
FIO03-C. fopen() を使用したファイル作成時に何らかの仮定をしない
FIO04-C. 入出力エラーを検出し、処理する
FIO05-C. 複数のファイル属性を使用してファイルを特定する
FIO06-C. 適切なパーミッションを持つファイルを作成する
FIO07-C. rewind() ではなく fseek() を使用する
FIO08-C. オープンしたままのファイルに対する remove() の呼び出しに注意する
FIO09-C. システム間でのバイナリデータ転送に注意する
FIO10-C. rename() 関数の使用に注意する
FIO11-C. fopen() のモード引数の指定は慎重に行う
FIO12-C. setbuf() ではなく setvbuf() を使用する
FIO13-C. 読み取った一文字以外は押し戻さない
FIO14-C. ファイルストリームにおけるテキストモードとバイナリモードの違いを理解する
FIO15-C. ファイル操作はセキュアなディレクトリで行われることを保証する
FIO16-C. ジェイル(監獄)を作成してファイルへのアクセスを制限する
FIO17-C. fread() を使用するときは、null 終端文字に依存しない
FIO18-C. fwrite() が書き込み操作を null 文字で終了すると想定しない

ルール

FIO30-C. ユーザからの入力を使って書式指定文字列を組み立てない
FIO31-C. 同一のファイルを同時に複数回開かない
FIO32-C. 通常ファイルに固有の操作をデバイスファイルに対して行わない
FIO33-C. 未定義の動作となる入出力のエラーを検出して処理する
FIO34-C. 文字入出力関数の返り値の取得には int 型を使用する
FIO35-C. sizeof(int) == sizeof(char) の場合、EOF およびファイルエラーの検出には feof() と ferror() を使用する
FIO36-C. fgets() が改行文字を読み取ると仮定しない
FIO37-C. 読み取られたデータが文字データであると思い込まない
FIO38-C. 入出力操作に FIlE オブジェクトのコピーを使用しない
FIO39-C. fflush 関数やファイル位置付け関数を呼び出さずにストリームへの入出力を交互に行わない
FIO40-C. fgets() が失敗したときは引数に渡した配列の内容をリセットする
FIO41-C. 副作用を持つストリーム引数を getc() または putc() に渡さない
FIO42-C. 不要になったファイルは正しくクローズする
FIO43-C. 共有ディレクトリに一時ファイルを作成しない
FIO44-C. fsetpos() には fgetpos() が返す値を使用する

例: FIO34-C: 文字入出力関数の返り値にはint型を使用する(1/5)

JPCERT CC®

文字入出力関数には、返り値として文字、エラー時にEOFを返す関数がある。エラー判定のためにEOFと比較する場合、返り値をchar型変数に格納してはいけない。正しく判別できなくなることがある。

```
int fgetc(FILE *stream);
int getchar(void);
    など
```

```
int fputc(int c, FILE *stream);
int putchar(int c);
    など
```

```
char c = getchar(void);
if (c != EOF){
    …….
}
```

返り値のint型データがchar型に切り捨てられる

比較演算のため、整数拡張でもういちどint型に変換される

Copyright© 2011 JPCERT/CC All rights reserved.

例: FIO34-C: 文字入出力関数の返り値にはint型を使用する(2/5)

JPCERT CC®

違反コード

```
char buf[BUFSIZ];
char c;
int i = 0;
while ( (c = getchar()) != '¥n' && c !=
EOF ) {
   if (i < BUFSIZ-1) { buf[i++] = c; }
}
buf[i] = '¥0'; /* null終端する */
```

入力文字が 0xFF だったとすると…

getchar()の返り値
0x000000FF (255)
c に代入される値
0xFF

比較演算(c != EOF)の際に
cは整数拡張され
0xFFFFFFFFに.

(char8ビット, int32ビット, EOFの値は(-1)の場合)

Copyright© 2011 JPCERT/CC All rights reserved.

例: FIO34-C: 文字入出力関数の返り値にはint型を使用する(3/5)

JPCERT CC®

適合コード

```
char buf[BUFSIZ];
int c;
int i = 0;
```

getchar()の返り値をそのまま保存する

```
while ( (c = getchar()) != '¥n'
              && !feof(stdin)
&& !ferror(stdin) ) {
    if(i<BUFSIZ-1){ buf[i++] = c; }
}
buf[i] = '¥0'; /* null終端する */
```

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

環境変数の操作や外部プログラム呼び出しに関するガイドライン

- getenv()が返す文字列を変更しない
- 同じ名前の複数の環境変数に注意
- 外部プログラムを呼び出す際は環境を無害化する

など

Copyright© 2011 JPCERT/CC All rights reserved.

10. 環境(ENV)

JPCERT CC®

レコメンデーション

ENV00-C. getenv() が返す文字列へのポインタを保存しない
ENV01-C. 環境変数のサイズについて勝手な想定をしない
ENV02-C. 同じ名前の複数の環境変数に注意する
ENV03-C. 外部プログラムを呼び出す際は環境を無害化する
ENV04-C. コマンドプロセッサが必要ない場合は system() を呼び出さない

ルール

ENV30-C. getenv() が返す文字列を変更しない
ENV31-C. 環境変数へのポインタを無効にするかもしれない操作の後で、そのポインタを参照しない
ENV32-C. atexit で登録したハンドラ関数は必ず return する

Copyright© 2011 JPCERT/CC All rights reserved.

例: ENV30-C: getenv()が返す文字列を変更しない (1/3)

getenv()が返す文字列を変更してはならない。変更を加えて使いたい場合、ローカルコピーを作成すること。

違反コード

```
void strtr(char *str, char orig, char rep) {
    while (*str != '¥0') {
          if (*str == orig) { *str = rep; }
         str++;
    }
}
/* ... */
char *env = getenv("TEST_ENV");
if (env == NULL) { /* エラー処理 */ }
strtr(env,'"', '_');
/* ... */
```

getenv()の返り値の文字列を直接変更している

Copyright© 2011 JPCERT/CC All rights reserved.

例: ENV30-C: getenv()が返す文字列を変更しない (2/3)

適合コード

```
void strtr(char *str, char orig, char rep) { /* ... */ }

const char *env;
char *copy_of_env;
env = getenv("TEST_ENV");
if (env == NULL) { /* エラー処理 */ }
copy_of_env = (char *)malloc(strlen(env)+1);
if(copy_of_env==NULL){ /* エラー処理 */ }
strcpy(copy_of_env, env);
strtr(env,'"', '_');
/* ... */
```

getenv()の返り値の文字列をcopy_of_envにコピーしてから変更している

Copyright© 2011 JPCERT/CC All rights reserved.

例: ENV30-C: getenv()が返す文字列を変更しない (3/3)

JPCERT CC®

適合コード(POSIXにおける環境の変更)

```
void strtr(char *str, char orig, char rep) { /* ... */ }

const char *env;
char *copy_of_env;
env = getenv("TEST_ENV");
if (env == NULL) { /* エラー処理 */ }
copy_of_env = strdup(env);
if(copy_of_env==NULL){ /* エラー処理 */ }
strtr(env,'"', '_');
if(setenv(“TEST_ENV”, copy_of_env, 1) != 0){
        /* エラー処理 */
}
```

POSIXでは、strdup() と setenv() を使ってこのように環境の変更が可能

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

シグナル処理に関するガイドライン

- シグナルハンドラでの処理
- シグナルハンドラの継続性
- `raise()`関数を再帰的に呼び出さない
- 「標準的な機能」にはシグナルを使わない

など

Copyright© 2011 JPCERT/CC All rights reserved.

11. シグナル(SIG)

JPCERT CC®

レコメンデーション

SIG00-C. 割り込み不可能なシグナルハンドラ によって処理されるシグナルをマスクする
SIG01-C. シグナルハンドラの継続性に関して処理系定義の詳細を理解する
SIG02-C. 標準的な機能を実装する際はシグナルの使用を避ける

ルール

SIG30-C. シグナルハンドラ内では非同期安全な関数のみを呼び出す
SIG31-C. シグナルハンドラ内で共有オブジェクトにアクセスしない
SIG32-C. シグナルハンドラ内から longjmp() を呼び出さない
SIG33-C. raise() 関数を再帰的に呼び出さない
SIG34-C. 割り込み可能なシグナルハンドラ内から signal() を呼び出さない

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

エラー処理に関するガイドライン

- エラー処理は一貫性のある方針で
- よりよいエラー検査をできる関数を使う
- `errno` の扱いについて

など

Copyright© 2011 JPCERT/CC All rights reserved.

12. エラー処理(ERR)

レコメンデーション

ERR00-C. エラー処理には一貫性のある方針を採用する
ERR01-C. errno ではなく ferror() を使って FILE ストリームエラーを検査する
ERR02-C. 正常終了時の値とエラーの値は別の手段で通知する
ERR03-C. TR24731-1 が定義する関数を呼び出す際は、実行時制約ハンドラを使用する
ERR04-C. プログラムの適切な終了方法を選択する
ERR05-C. アプリケーション非依存なコードではエラー検知のみ行ない、エラー処理は行わない
ERR06-C. assert() と abort() の終了動作を理解する
ERR07-C. よりよいエラー検査を行える関数を使用する

ルール

ERR30-C. 関数を呼び出す前に errno をゼロに初期化し、関数の異常終了時にのみ errno を参照する
ERR31-C. errno を再定義しない
ERR32-C. errno の未規定の値を参照しない

Copyright© 2011 JPCERT/CC All rights reserved.

API に関するガイドライン

関数のなかで引数を検証する
関連する関数の機能と
　　インタフェースに一貫性を持たせる
型の安全性を徹底する

など

"This section is under construction."

Copyright© 2011 JPCERT/CC All rights reserved.

レコメンデーション

API00-C.関数のなかで引数を検証する

API03-C.関連する関数の間ではインタフェースと機能に一貫性を持たせる

API04-C.一貫性があり使いやすいエラー検査方法を提供する

API07-C.型の安全性を徹底する

API08-C.ヘッダのプロトタイプ宣言が誤解釈されるのを防ぐ

API09-C.互換性のある値には同じ型を使用する

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

並行性に関するガイドライン (POSIX セクションから派生)

- スレッド間の競合状態を避ける
- 同期プリミティブは同一抽象レベルで
- 複数スレッドのアクセスはミューテックスで保護

など

Copyright© 2011 JPCERT/CC All rights reserved.

14. 並行性(CON)

JPCERT CC®

Recommendations

CON00-C. 複数のスレッド間の競合状態を避ける

CON01-C. 同期用プリミティブの取得と解放は、同じ翻訳単位内の同一抽象レベルで行う

Rules

CON31-C. 他のスレッドのミューテックスをアンロックしたり破壊したりしない

CON32-C. 複数スレッドによる同一データへのアクセスは、ミューテックスで保護する

CON33-C. ライブラリ関数を使うときの競合状態を避ける

CON34-C. 複数のスレッドで共有するオブジェクトは適切な記憶域期間で宣言する

CON35-C. Avoid deadlock by locking in predefined order

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

他のカテゴリに収まらないもの

- 乱数生成に関する注意
- 文字の扱い
- 未定義の動作に依存しない
- デッドコード
- 関数の返り値は適切な型と比較する

などなど

Copyright© 2011 JPCERT/CC All rights reserved.

49. 雑則(MSC)

JPCERT CC®

レコメンデーション

MSC00-C.高い警告レベルで警告を出さずにコンパイルする
MSC01-C.論理的な完全性を追求する
MSC04-C.コメントの記法には一貫性を持たせ読みやすくする
MSC05-C.time_t 型の値を直接操作しない
MSC06-C.機密データを扱う場合はコンパイラの最適化に注意する
MSC07-C.デッドコードを検出して削除する
MSC09-C.文字の符号化 - 安全対策のために ASCII のサブセットを使用する
MSC10-C.文字の符号化 - UTF8 に関連する問題
MSC11-C.診断テストはアサートを使って組み込む
MSC12-C.プログラムに対して作用しないコー ドを検出して削除する
MSC13-C.使用されない値を検出して削除する
MSC14-C.必要もなくプラットフォーム依存のコードを書かない
MSC15-C.未定義の動作に依存しない
MSC16-C.関数ポインタの暗号化を検討する
MSC17-C.case 句に関連付けられた一連の文は break 文で終了する
MSC18-C.プログラムコードの中でパスワードなどの機密情報を扱うときは注意する
MSC19-C.配列を返す関数は、NULL 値ではなく空の配列を返すこと
MSC20-C.複雑なブロックに制御を渡す際に switch 文を使用しない
MSC21-C.ループの終了条件には不等式を用いる

ルール

MSC30-C.疑似乱数の生成に rand() 関数を使用しない
MSC31-C.関数の返り値は必ず適切な型と比較する
MSC32-C.乱数生成器は適切なシード値を与えて使う

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

POSIX環境に関するガイドライン
「付録」扱い

ファイル操作とシンボリックリンクの存在
最小権限の原則(setuidなどの使い方)
権限の破棄は正しい順序で行う

など

Copyright© 2011 JPCERT/CC All rights reserved.

50. POSIX(POS)

JPCERT CC®

レコメンデーション

POS01-C.ファイルを操作するときにはリンクが存在するかどうかを検査する

POS02-C.最小権限の原則に従う

POS03-C.volatile を同期用プリミティブとして使用しない

ルール

POS30-C.readlink() 関数を正しく使用する

POS33-C.vfork() は使用しない

POS34-C.自動変数へのポインタを引数として putenv() を呼び出さない

POS35-C.シンボリックリンクの有無のチェック時の競合状態を避ける

POS36-C.権限は正しい順序で破棄する

POS37-C.権限の破棄は確実に行う

Copyright© 2011 JPCERT/CC All rights reserved.

今日お話する内容

JPCERT CC®

セキュアコーディングスタンダード概論

セキュアコーディングスタンダードって?
中身を見てみよう
今後の展開

セキュアコーディングスタンダード各論

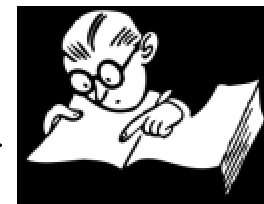
内容をより詳しく見ていきます

セキュアコーディングスタンダード活用

どんな活用ができるか、そのアイディアを語ります

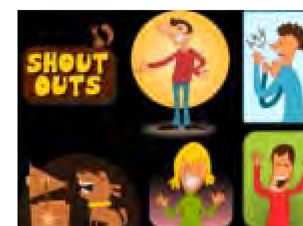
Copyright© 2011 JPCERT/CC All rights reserved.

- セキュアコーディングの学習に



- 開発チーム内の認識共有

- 製品のセキュリティ関連仕様の明確化



Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

個人単位でセキュアコーディングを学ぶ教材として使う

- 「セキュアコーディングってなに?」
- 「こんなトピック知ってる?」

社内での研修や勉強会の教材に

- 違反コード・適合コードでクイズ形式

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

コード解析ツールを活用するための補助資料
検出結果を理解するために

ツールの説明よりも分かりやすく
別の観点での説明

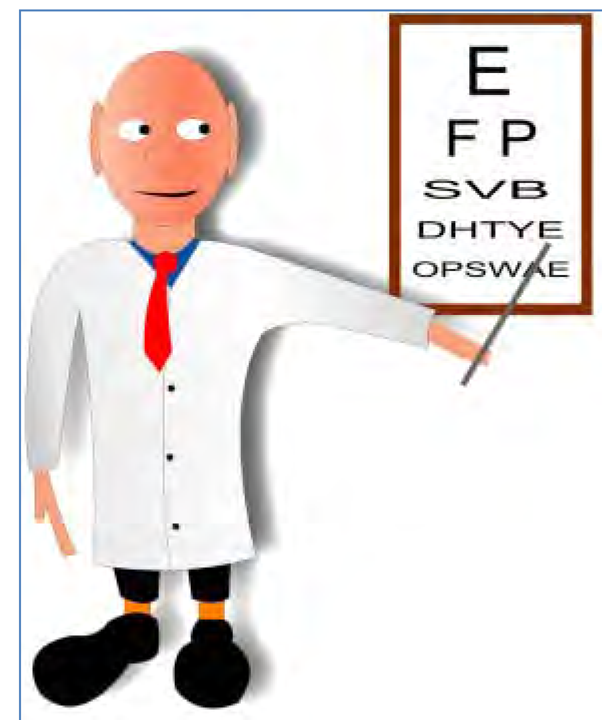
Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

コード解析ツールを活用するための補助資料
検出結果の対応を検討するために

どのように修正すべきなの?
ホントに修正すべきなの?

セカンドオピニオンとして...



Copyright© 2011 JPCERT/CC All rights reserved.

活用: 開発チーム内の認識共有(1)

JPCERT CC®

コーディングの良し悪しの判断基準

セキュリティ確保のための目標を具体化

- 「ルールXXとYYには準拠しよう！」

セキュリティ品質向上の手段として活用

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

セキュリティ確保のためのコーディング規約

- 既存の内容に不足がないかチェック
- 命名規則やコーディングスタイルなどは独自に決める必要がある

こういうものをつくるときに役に立ちます

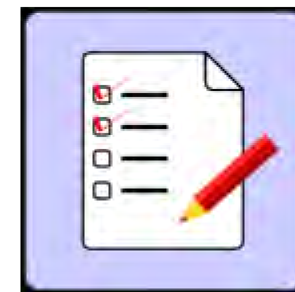
Sony
製品確保に関する基準 STM-0117
http://www.sony.co.jp/SonyInfo/procurementinfo/software/rule.html

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

品質チェックの具体的な項目の洗い出しに

入力値チェックはきちんとやってるか
非推奨な関数使ってないか
可搬性の問題はないか
……



Copyright© 2011 JPCERT/CC All rights reserved.

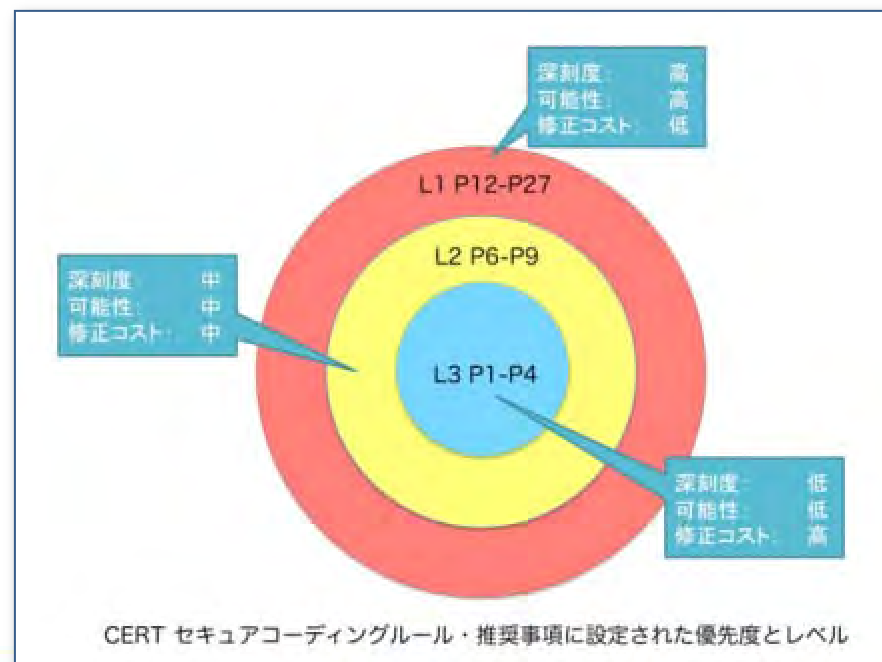
JPCERT CC®

要求仕様として提示

- コーディング規約XXを遵守すること
- レベルL1のルールを遵守すること
- レベルL2までのルールを全て遵守すること

など

あいまいだったセキュリティ面の仕様の内容について、より明確な基準で議論できる



CERT セキュアコーディングルール・推奨事項に設定された優先度とレベル

Copyright© 2011 JPCERT/CC All rights reserved.

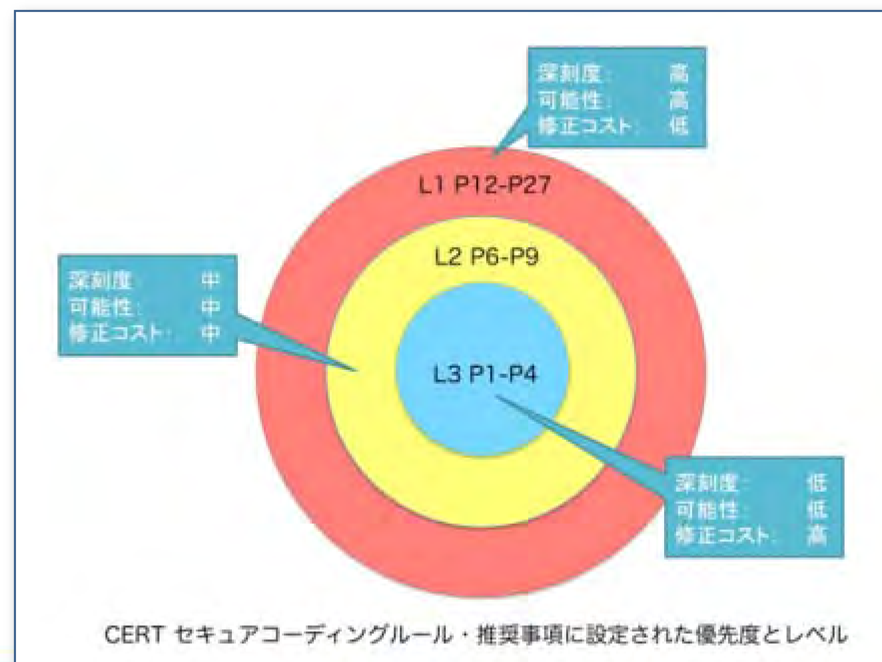
JPCERT CC®

実装仕様として提示

- コーディング規約XXを遵守**してます**
- レベルL1のルールを遵守**してます**
- レベルL2までのルールを全て遵守**してます**

など

あいまいだったセキュリティ面の仕様の内容について、より明確な基準で議論できる



CERT セキュアコーディングルール・推奨事項に設定された優先度とレベル

Copyright© 2011 JPCERT/CC All rights reserved.

JPCERT CC®

セキュアコーディングスタンダード活用してね!

日本語版は JPCERT/CC のwebサイトにあるよ
(https://www.jpcert.or.jp/sc-rules/)

書籍もあるよ

今後の動向に乞うご期待!

Copyright© 2011 JPCERT/CC All rights reserved.

関連リソース

JPCERT CC®

CERT C Secure Coding Standard 日本語版
(https://www.jpcert.or.jp/sc-rules/)

CERT Secure Coding Standards
(https://www.securecoding.cert.org/)

CERT/CC Secure Coding
(https://www.cert.org/secure-coding/)

CERT Rose Checkers
(http://sourceforge.net/projects/rosecheckers/)

Copyright© 2011 JPCERT/CC All rights reserved.

関連リソース

【改訂版】組込みソフトウェア開発向けコーディング作法ガイド［C言語版］
(http://www.seshop.com/product/detail/7950/)

MISRA-C:2004　ガイドブック
(http://www.sessame.jp/workinggroup/WorkingGroup3/MISRA-C_GuigeBook2004.htm)

組込み現場の「C」プログラミング標準コーディングガイドライン
(http://gihyo.jp/book/2007/978-4-7741-3254-9)

Copyright© 2011 JPCERT/CC All rights reserved.

参考情報

- JPCERT/CC セキュアコーディングのページ
  https://www.jpcert.or.jp/securecoding.html
- セキュアコーディングセミナに関する情報
  https://www.jpcert.or.jp/event/
- セキュアコーディングセミナの講義資料のダウンロード
  https://www.jpcert.or.jp/research/materials.html

問い合わせ

JPCERTコーディネーションセンター
セキュアコーディング担当
Email：secure-coding@jpcert.or.jp

Copyright© 2011 JPCERT/CC All rights reserved.