

**GLOBAL  
INFORMATION  
ASSURANCE  
CERTIFICATION**



**CANDIDATE BULLETIN**

**GIAC  
Secure Software  
Programmer  
(GSSP)**

***Secure Programming Skills in C***

**GIAC Secure Software Programmer (GSSP)  
C 言語 受験要綱**

**SANS**

In Cooperation with the SANS Software Security Institute



# GIAC Secure Software Programmer (GSSP) C 言語 受験要綱

## 目 次

A. 序章 .....	2
GIAC 認定試験 .....	2
B. GIAC SECURE SOFTWARE PROGRAMMER (GSSP) 試験 .....	2
C. 受験資格 .....	3
D. GIAC 倫理規程 .....	3
E. 試験情報 .....	4
受験申込み期限・試験日程 .....	4
受験料 .....	4
受験申込み手続き .....	4
出題内容 .....	5
試験時間 .....	5
試験当日 .....	5
受験規則 .....	6
成績 .....	6
認定 .....	7
再受験 .....	7
F. サンプル問題と解答 .....	8
付録 A : コンテンツブループリント .....	12
C 言語におけるセキュアプログラミングスキル .....	12
付録 B:参考文献 .....	29
ソフトウェアセキュリティに関する参考書 .....	29
ソフトウェアセキュリティに関する Web サイトや Podcast .....	30

## A. 序章

開発者やプログラマーがより安全なコーディングスキルを身につけない限り、政府や企業、教育機関は、ソフトウェア脆弱性対応という終わりなき戦いを強いられる運命にあります。そこで、セキュアプログラミングスキルおよび知識を促進する各イニシアチブが進行中です。シマンテック、オラクル、マイクロソフト、ほか数社のソフトウェア企業は、自社プログラマー向けに短期研修を行っていますし、SPI Dynamics や Fortify Technology は、大学と連携してプログラミングコースの受講生にリアルタイムフィードバックシステムを提供しています。また、多くの大学がセキュアプログラミングに関するオンラインコースを設けています。しかし、たとえこうしたイニシアチブの全てが功を奏しているとしても、その影響が、既に実務に就いている、もしくは今後 5 年間に開発者やプログラマーになると想定される 150 万人のうちの 2%にも及んでいるかどうか疑わしいのが現状です。

このような状況に対応するため、SANS Institute は Global Information Assurance Certification(GIAC)を通じて、ソフトウェアのプログラマーおよび開発者向けの試験認定制度を整備し、セキュリティ脆弱性につながる一般的なプログラムエラーの確認や修正に要する技術の習熟度と専門知識の有無について、信頼できる基準の策定を行ってきました。

SANS は、情報セキュリティのトレーニングおよび認定試験において、世界で最も信頼できる最大の組織です。SANS による様々な情報セキュリティ調査研究結果の膨大な資料は、無償で入手することができます。また、インターネットの早期警戒システムである Internet Storm Center の運営も行っています。SANS は、盛んに弱点をつかれる恐れのある最大の脅威から、アプリケーションやシステム、ネットワークの防御に要する実践的な技術を習得することができる集中トレーニングを提供しています。

### GIAC 認定試験

GIAC は Global Information Assurance Certification の略であり、1999 年、IT セキュリティの専門家や開発者の実質的なスキルの有効性を証明するべく設立されました。コンピュータ、ネットワーク、アプリケーション、そしてソフトウェアのセキュリティに関する主要分野の実践的な知識とスキルの保持者の認定を目的としています。この認定資格は、情報セキュリティおよび安全なソフトウェア開発の実務に伴う様々な責任を担うに相応しいものです。GIAC は、総合的な知識よりも、具体的に特定された知識を推し量るという点に独自性があります。

## B. GIAC SECURE SOFTWARE PROGRAMMER (GSSP) 試験

GIAC Secure Software Programmer (GSSP) は、ほとんどのセキュリティ問題につながる一般的なプログラムエラーの対処に必要な知識とスキルの習得者を認定します。対応言語は以下のとおりです。

- Java/JEE
- PHP
- C
- Perl

・ C++

・ ASP .NET and .NET

※ 「Java/JEE」「C」以外の言語については開発中

この試験は、まず第一に開発やコードの維持に携わるプログラマー／開発者を対象としています。すなわち、ソフトウェアコードの検証／監査、品質保証や検査のスキルも重要となる業務の担当者です。試験では、脆弱性につながる一般的なエラーに焦点を当てます。日々のコーディング作業に直接関わる安全なプログラム規則のみならず、各プログラム言語の遂行課題の解決も目標としています。

机上の知識を測るのではなく、コーディングエラーの認知力、妥当な規則の適用力、安全なコーディングの核を構成できる提案力、といった実践的な能力を測る試験です。試験問題の多くが、コードサンプルからのエラーの発見と除去方法の決定を問うものになります。

### C. 受験資格

GSSP の受験資格は特にありません。

### D. GIAC 倫理規程

情報セキュリティの専門家と開発者の影響範囲と責任領域は多岐にわたります。提供するサービスは、組織の成功を左右するものであり、IT およびソフトウェア開発業界のセキュリティ全体に対する姿勢に関わる重要なものです。そのような責任は、情報セキュリティおよび安全なアプリケーション開発規則の実践倫理基準をクリアした認定資格者に対する重大な期待の下に位置づけられます。

GIAC 認定は、獲得・維持に値する荣誉であると認知されています。GIAC 認定資格者は、当該倫理規程の提唱、順守、支持を誓約します。

当該規定のいかなる原則に対して故意に違反する GIAC 認定資格者は、GIAC より懲戒処分を受けます。

#### 社会に対する責任

私は、コミュニティのセキュリティや福祉に関係する決定を下すに当たり、相応の責任を負います。

私は、コミュニティ、私のプロフェッショナルとしての評価、または情報セキュリティの原則に悪影響を与えるような非倫理的あるいは違法な集団と関係を持ちません。

#### 認定に対する責任

私は、GIAC 認定手続きに関する機密情報を他に漏らしません。

私は、私の認定および認定に関連する物や情報（認定証やロゴといったもの）を、GIAC 認定資格者である私自身を象徴するため以外、いかなる個人または団体の用にも供しません。

## 雇用主に対する責任

私は、私の認定資格と地位に対する期待に見合うレベルの高いサービスを提供するよう努めます。

私は、知り得た機密情報の守秘義務を負います。

私は、リスクマネジメントを実践することにより、IT ソリューションの機密性、完全性、可用性に対するリスクを最小限に抑えます。

## 自身に対する責任

私は、利害関係の衝突を回避いたします。

私は、私の立場上保持するいかなる情報および特権も悪用いたしません。

私は、私の技能、コミュニティにおける職務、雇用主、同僚について偽装いたしません。

## E. 試験情報

### 受験申込み期限・試験日程

GSSP 試験の受付は年中行っています。直近の実施情報はこちらです。

<http://www.sans.org/gssp/>

申し込みは、希望試験日の 30 日前までに済ませてください。

### 受験料

日本国内で開催される GSSP 試験の受験料は 57,000 円です。

支払い方法は、口座振込みのほか、Visa、マスターカード、JCB、American Express が利用できます。日本国内開催の受験料は、すべて円建てのお支払いとなります。受験料はいかなる理由があろうとも払い戻しできません。試験運営事務局の入金確認が済んだ時点で、お申込み受付の完了となります。

### 受験申込み手続き

1. 申込みに先立って受講要領に目を通し、記載内容に従ってください。手続きに失敗すると申込みができない場合がありますのでご注意ください。
2. 受験申込みサイトで申込みを完了させます。登録項目の全てを埋めてください。ここで入力された情報が、受験者との連絡に必要となります。手続き迅速化のため、郵便番号、電話番号、E メールアドレス全てに漏れない完全な情報を提供してください。申込みおよび認定の手続き期間中は、常に最新情報の受信を可能にするため、SANS に提供する受験者の属性情報は最新の状態であるよう留意してください。

以下の理由により、GSSP 受験の申込みが拒否されたり、GIAC 認定が取り消されたりすることがあるので注意してください。

- ・不備のある申込み
- ・登録情報の改ざんまたは虚偽の登録
- ・GIAC 倫理規程違反による罰則の適用

3. 試験日の約 3 週間前までに、GSSP 試験実施事務局より各受験者に受験票を送付いたします。試験当日、必ず持参してください。

### 出題内容

多岐選択式問題が 100 問出題されます。

2 つ以上の選択肢を回答する問題もあり、その旨明記されています。

各試験に数種類のバージョンがありますが、全受験者に間違いなく公平を期すため、実施バージョンによって合格可能性に偏りはありません。上記「試験沿革」の項にあるとおり、専門家によって識別された全試験問題の難易度レベルが、難易度比較可能な試験内容に反映されています。

### 試験時間

試験の制限時間は 6 時間です。

終了した方は、随時退席可能です。

### 試験当日

試験開始の 10 分前に受験に関する注意事項を説明します。試験開始時刻の 10 分前には試験会場に入場してください。

また、必ず顔写真入りの身分証明書を持参してください。

身分証明書となるもの：

運転免許証

パスポート

その他政府公認の身分証

身分証明書とならないもの：

スポーツジムの会員証

店舗の会員証

学生証

クレジットカード

顔写真のない証明書

※試験場への入室前に、本人確認をさせていただきます。

鉛筆、メモ用紙、回答用紙、問題冊子は試験会場で配布いたします。これらは全て試験室より持ち出し禁止となります。

## 受験規則

### 1. 試験の保証事項と手順

- いかなる書籍、書類、その他参考資料も試験室へは持ち込めません。
- 電話、ノート PC、カメラ、信号を発する媒体、ポケベル、アラーム機器、計算機、および録音／再生機器（iPod や mp3 プレイヤーを含む）など、いかなる電子機器も試験室へ持ち込めません。これらのものは試験会場に持参しないようにしてください。
- 音を遮るための耳栓は持ち込み可能です。
- 試験を終えた受験者は試験室より退出してください。
- 受験しないいかなる同伴者の入場も認めません（親、子供、配偶者、友人、介助動物以外のペットほか、いかなる人物や動物も不可）。
- 試験に要した道具、資料や全てのメモは試験室より持ち出せません。試験で使用した物は退出の前に全て提出してください。
- 試験施行中は、他の受験者と議論したり、参考資料や試験情報についていかなる情報共有も行わないでください。試験室退出後の情報共有も認められていません。
- いかなる理由があっても、試験問題のコピーはできません。
- 飲食物は指定された場所でのみとることができます。
- 試験施行中は、試験に関するいかなる質問も受け付けません。試験監督の説明をよく注意して聴いてください。
- 試験は、受験票記載の日時にのみ受験することが可能です。
- ドレスコードはビジネス・カジュアルです。
- 空調は可能な限り調整いたしますが、念のため寒暖に備えられる服装で臨んでください。

### 2. 参考資料

適正な試験運営のため、試験監督より配布された試験資料以外の物へは何も書き込まないでください。参考資料は試験室へ持ち込めません。受験者の私物の持参は最小限にとどめてください。

### 3. 解答用紙

解答用紙には、受験者の氏名およびその他指定の必要事項を記入してください。解答はすべてこの用紙に記入してください。解答が終了したら、試験監督が受験者の試験関連物の回収を行うまで座席で待機してください。問題冊子に記入された解答は採点対象となりませんので、解答を問題冊子から解答用紙に書き写す作業は制限時間内に終わってください。

## 成績

試験結果は、試験日より 6 週間後に米国より郵送いたします。合否のほか、総得点と各試験分野ごとの得点内訳が記載されます。電話や E メール、FAX による照会はできません。

## 認定

合格者は **GIAC Secure Software Programmer (GSSP)** 認定を付与されます。選択した開発言語に応じた認定内容となります。例えば、**C** 言語の試験に合格したプログラマーは **GSSP-C** 認定取得者となります。

**GSSP** 認定は **4** 年間有効です。有効期限最後の **1** 年間、更新試験受験資格が与えられます。更新試験の受験申込み受付期間は **この 1 年間**です。更新試験に合格しなければ、認定資格失効となります。

## 再受験

不合格の場合、試験日より **4** ヶ月の待機期間後より再度同様の申込み手続きが可能となり、受験できます。受験料は同額です。

## F. サンプル問題と解答

### サンプル問題

1) 次の関数においてセキュアでない点はどれでしょうか。

```
char *readbuf(char buf[]) {  
    printf("Enter input >");  
    return(gets(buf));  
}
```

- A. この関数が**gets()**を呼び出していること。
- B. 配列**buf**のサイズが関数宣言で明示的に指定されていないこと。
- C. **printf()**の呼び出しが、プログラムスタックをオーバーフローさせるデータを使用してリターンアドレスを変更しようとする攻撃に対して脆弱であること。
- D. なし。セキュアなコードである。

2) 下記のコードに示す加算演算の結果の型を選択してください。ただし2の補数表現であり、**char**は8ビット、**int**は32ビットであるとします。

```
unsigned char c = 5;  
unsigned char d = 7;  
if (c + d > UCHAR_MAX) abort();
```

- A. **unsigned char**
- B. **signed char**
- C. **signed int**
- D. **unsigned int**

3) 次の整数演算のうち、整数のオーバーフローが発生しないものはどれでしょうか。

- A. 乗算
- B. 除算
- C. 単項否定
- D. 右シフト

4) 次の演算のうち、式**E1 >> E2**において**E1**が有符号型であり、負の値を持つ場合に実行されるものはどれでしょうか。

- A. 未定義

- B. 算術シフト
- C. 論理シフト
- D. 処理系依存

5) arcインジェクション攻撃 (return-to-libc攻撃とも呼ばれる) が危険である理由として、正しい記述を選択してください。

- A. 書き込み可能なページに実行不可と指定しても攻撃を回避できないため。
- B. バッファオーバーフローを引き起こす可能性があるため。
- C. あらゆるメモリセグメントで、脆弱性コードを実行できるため。
- D. 挿入されたコードにより、通常の入力チェック手段を回避できるため。

6) 下記のコードにおいて、オブジェクトの境界外への書き込みを許可してしまうおそれがあるのは何行目でしょうか。

```
1. #include <stdio.h>
2. #include <string.h>
3. void usage(char *ptCommand) {
4.     char usage [1023];
5.     snprintf(usage, 1023, "Usage: %s <target>¥n", ptCommand);
6.     printf(usage);
7. }
8. int main(int argc, char * argv[]) {
9.     if (argc < 2)
10.         usage(argv[0]);
11. }
```

- A. 5行目
- B. 6行目
- C. 9行目
- D. 10行目

7) 競合状態に起因する脆弱性を解消するには、競合するウィンドウをどのようにすべきでしょうか。

- A. できるだけ小さくする。
- B. 閉じる。
- C. クリティカルセクションにする。

D. 相互排他にする。

8) 下記のCの関数には、どのようなコーディング上の問題点が存在するでしょうか。

```
char *copybuf(char str[]) {
    char *p;
    p = malloc(strlen(str)+1);
    (void) strcpy(p, str);
    return(p);
}
```

- A. `strncpy`でなく`strcpy()`を呼び出している。
- B. `malloc()`の戻り値をチェックしていない。
- C. `malloc`の型を`char *`に型変換していない。
- D. なし。正しい関数である。

9) 下記のプログラムにおける`size`の値と等価であるものを選択してください。

```
void func(char s[]) {
    size_t size = sizeof(s) / sizeof(s[0]);
}
int main(void) {
    char str[] = "0123456789";
    func(str);
}
```

- A. `sizeof("0123456789")`
- B. `sizeof("0123456789") + 1`
- C. `sizeof(char *)`
- D. `sizeof("0123456789") / sizeof(char)`

10) `p`を`T`型の定数ポインタであると宣言するステートメントを選択してください。

- A. `T * const p;`
- B. `T const * p;`
- C. `p * const T;`
- D. `p const * T;`

## 解答

- 1) A
- 2) C
- 3) D
- 4) D
- 5) A
- 6) B
- 7) D
- 8) B
- 9) C
- 10) A

## 付録A：コンテンツブループリント

### C言語におけるセキュアプログラミングスキル

本資料は、必要に応じて更新が行われる可能性があります。最新版は[www.sans-ssi.org](http://www.sans-ssi.org)よりダウンロードしてください。ご意見ご要望は[spa@sans.org](mailto:spa@sans.org)までお送りください。

# GSSP（GIACセキュアソフトウェアプログラマ）

## 01 C セキュアコーディングにおける課題、スキル、および知識

本資料では、C言語でのコーディングにおける一般的な課題を挙げ、これらの課題をセキュアに遂行するための規則、推奨事項、およびガイドラインを説明する。

### 課題1:

Cプログラムは、実行環境とセキュアに対話しなければならない。プログラムは、実行環境からの入力を受け入れ、コマンドライン引数や環境変数などを含むこれらの入力を、適切に検証および処理しなければならない。またプログラムでは、外部プログラムをセキュアな方法で呼び出し可能である必要がある。

#### 01.1.1 入力の検証

プログラマは、実行環境におけるあらゆる状況からの入力を正しくデコード、正規化、検証した上で、セキュアに処理しなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. すべての入力ソースは検証されなければならない。信頼できないあらゆるソースからのデータは完全に指定されるべきであり、これらの仕様に照らし合わせてデータが検証される必要がある。すべての入力のデータが、予期されているデータ型に該当し、構文のおよび意味的に有効であることを確認する。
2. プログラムに対し、可能な限り最も強固な入力検証機能を使用する（例：選択メニューを用いた間接的な選択操作など）。
3. クライアント/サーバーアーキテクチャまたは3層構造アーキテクチャでは、クライアントがスプーフィングや改竄を受ける可能性があることを想定する。
4. プログラムに対して短すぎる入力と長すぎる入力はすべて拒否する。
5. 妥当な範囲制限を設定し、強制する。最小有効値を下回る数値入力、および最大有効値を上回る数値入力はすべて拒否する。
6. 文字データ入力を受け入れて自動的に変換する関数は、すべての有効な入力を正しく処理できない限りは使用しない。
7. どのような環境変数でも、取り扱う際にまず疑ってみる。環境変数のサイズ、値または存在に関

する仮定をしない。

- あらゆる入力を正規形に変換し、変換処理（適切なデコードを含む）後の入力を検証する。これは、URLのように、パス名やファイル名など、この時点以降別のシーケンスに変換される可能性のある文字シーケンスを含む入力文字列に適用される。
- 入力が必ず1回だけデコードされるようにする。

#### スキル:

- バッファオーバーフロー、ディレクトリトラバーサル、正規化エラー、XML検証の欠落に起因する脆弱性の防止。
- スプーフィング攻撃の防止。

### 01.1.2 データのサニタイズ

プログラマは、データを正しくサニタイズしてエンコードし、攻撃者が直接の影響を与えないよう、サブシステムや外部プロセスにデータを渡さなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

- ユーザ操作による入力を複雑なサブシステム(コマンドインタプリタなど)に渡すことで起こるコマンドインジェクション攻撃を防ぐ。このような呼び出しでユーザ入力を必要とする場合は、サブシステムに渡すことができる文字列の形式を詳細に指定しておき、呼び出し前にこの仕様に突き合わせて入力文字列をサニタイズする。
- 組み込み関数を使用できる場合、メタ文字はすべて、組み込み関数を使用してエスケープする。
- リレーショナルデータベースを用いるインタフェースでは、ホワイトリストに指定されていないSQLメタ文字が含まれているユーザ入力はすべて拒否する。アプリケーションが受け入れる必要のあるSQLメタ文字(名前での一重引用符の使用など)はすべてエスケープする。SQL文は常にPREPAREステートメントを使用して作成する。

#### スキル:

- OSコマンドインジェクション攻撃、クロスサイトスクリプティング(XSS)攻撃、SQLインジェクション攻撃、CRLFインジェクション攻撃の防止。

### 01.1.3 呼び出し

プログラマは、外部プログラムを呼び出す際には、元のプログラム実行環境のコントロール権を、完全に維持しなければならない。また、正しいプログラムが呼び出され、必要な引数だけが使用されていることを検証する。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 外部プログラムの呼び出し前に実行環境をサニタイズする。プログラムに必要なもの以外のすべての環境変数をパージし、デフォルト値として安全な値を設定することを検討する。
2. 同名の環境変数が複数存在する可能性や、利用する環境変数に信頼できない値が含まれている可能性を見込んでおく。
3. 管理者のコントロール下でない場所や信頼できない場所を、検索パスに指定できないようにする。
4. 呼び出すコンポーネントに渡されるすべての引数を完全に管理できるようにし、引数の区切り文字や引用符文字を削除しておく。
5. 汎用コマンドインタプリタの機能が不要な場合は、プログラム実行のために汎用コマンドインタプリタを呼び出す操作を避ける。これは、たとえば`system()`の代わりに`exec()`を使用することで可能となる。
6. 名前付きパイプなど、外部からアクセスできるリソースを使用して呼び出しが実行される場合、これらのリソースでは、受け取ったリクエストの完全性と呼び出し元を検証する必要がある。
7. 過度の権限が設定されたプロセスを生成しない。
8. 機密データをコマンド引数としてプログラムに渡さない。このように操作すると、`ps`コマンドの実行によりユーザがこの機密データを確認できてしまう可能性がある。

#### **スキル:**

1. 信頼できない検索パスや呼び出し元検証エラーに起因する脆弱性の防止。
2. 引数インジェクション攻撃の防止。

## 課題2:

Cプログラマは、動的に割り当てるメモリなど、動的割り当てリソースを管理できなければならない。

### 01.2.1 メモリ管理

プログラマは、メモリの割り当て、管理、解放を適切に実施できなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 動的に割り当てられたメモリのみを解放する。
2. 同一レベルの抽象化や、参照カウントなどで、同じモジュール内でメモリの割り当てと解放を行うことで、メモリを複数回解放しないようにする。
3. 解放されたメモリにはアクセスしない。
4. 0バイトのメモリを割り当てないようにする。
5. メモリ割り当て関数のサイズ引数が有効であることを検証する。
6. 動的に割り当てられたメモリを1回だけ解放する。
7. 割り当てが解放されたメモリを指していたポインタは、メモリ解放直後にNULLに設定する。
8. NULLを指している可能性のあるポインタにアクセスする前に、ポインタ値がNULLであるかどうかを検査する。
9. 文字幅などのデータ型の違い(charとwchar\_t)や、文字列のNULL終端バイトの存在を考慮した上で、正しいバッファサイズを計算する。
10. バッファオーバーフローを防ぐため、動的に割り当てたメモリのサイズを追跡する。
11. プログラム中でめったに実行されない命令や、メモリ関連エラーのエラー処理コードを慎重に評価する。
12. DoS(サービス拒否)攻撃を阻止するため、リソース割り当て量を決め、強制する。
13. 重要な情報を、必要以上に長い期間にわたりメモリ内に保持しない。
  - a. たとえばメモリ解放の前に、メモリに格納されているが今後は不要となる重要な情報は、すべて消去または他の値で上書きする。
  - b. コンパイラの最適化により実行コードからこれらのメモリ消去命令が除去されないように、volatileポインタを使用する。
  - c. すべてのメモリオブジェクト、特にシリアライズされたり、信頼できないエンドポイントに伝送されたりする可能性のあるメモリオブジェクトは、完全に初期化する。
  - d. 重要な情報が格納されているメモリに対してrealloc()を実行しない。

#### スキル:

1. バッファオーバーフロー、メモリリーク、同一メモリの解放操作の繰り返し、既に解放されているメモリへの書き込みに起因する脆弱性の防止。
2. ヒープインスペクション攻撃とDoS(サービス拒否)攻撃の防止。

### 課題3:

Cプログラマは、ストリームおよびファイルからの入出力を実行でき、またオペレーティングシステムを操作してファイルシステムを管理できなければならない。

#### 01.3.1 フォーマット出力

プログラマは、フォーマット出力関数への入力を慎重にコントロールしなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. フォーマット入出力関数(`printf()`、`fprintf()`、`sprintf()`など)のフォーマットパラメータにユーザ操作による入力を渡してはならない。
2. すべての各国語文字列に、必要なフォーマット指定子を必ず含めるようにする。
3. レコードなどの整形形式出力では、フィールドに区切り文字が含まれていないようにする。

#### スキル:

1. フォーマット文字列による脆弱性と、バッファオーバーフローに起因する脆弱性の防止。
2. メモリインスペクション、情報漏洩、DoS(サービス拒否)攻撃の防止。

#### 01.3.2 ストリームおよびファイル入出力

プログラマは、適切な権限とアクセス権をもって、正しいファイルとストリームを作成、変更、および削除する必要がある。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. ユーザによる変更が可能なファイルへの書き込みを実行する前に、プロセスの特権を削除する。一般に、すべてのファイル入出力操作は低い権限で実行する。
2. 信頼できないソースから与えられたファイルの名前を正規化する。
3. 不正なアクセスを阻止するため、適切なアクセス権を設定したファイル/ディレクトリ構造を作成する。
4. ファイル作成時に、作成するファイルが既に存在しているかどうかを確認する。
5. ファイルの識別の際には、ファイル名をあてにせず、複数のファイル属性を使用してファイルを識別する。たとえば、ファイル所有権や作成時刻を比較する。
6. 信頼できないプロセスにファイルディスクリプタがリークされていないか検証する。
7. TOCTOU(チェック時から使用時まで)競合状態を回避する。
8. ファイルへのアクセスを制限するため、ディレクトリ制限操作(`chroot()`や`jail()`など)を採用する。
9. トラバーサル攻撃を阻止するため、ファイル名には可能な限り絶対パスを使用する。
10. 入力サイズを制限しない関数(`gets()`など)の使用を避ける。
11. 文字入出力関数から戻される値がEOFと比較される場合は、この値を`char`に変換しない。EOFを

検出するには`feof()`を、ファイルエラーを検出するには`ferror()`をそれぞれ使用する。

12. 文字列の読み取りでは、改行文字が読み取られること、または文字データが読み取られることを仮定しない。

#### スキル:

1. 競合状態(TOCTOUを含む)とセキュアではないアクセス権に起因する脆弱性の防止。
2. ディレクトリトラバーサル攻撃、シンボリックリンク攻撃、およびハードリンク攻撃の防止。

### 01.3.3 一時ファイル

プログラマは、一時ファイルを適切に作成、保護、削除できなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 共有ディレクトリでの一時ファイルの作成を避ける。共有ディレクトリに一時ファイルを作成しなければならない場合は、`mkstemp()`関数またはTR 24731-1 `tmpfile_s()`関数を使用する。
2. `tmpnam()`、`tempnam()`、`mktemp()`など競合状態を引き起こす関数や、`tmpfile()`など予測可能な名前を生成する関数の使用を避ける。

#### スキル:

1. 競合状態(TOCTOUを含む)とセキュアではないアクセス権に起因する脆弱性の防止。
2. ディレクトリトラバーサル攻撃、シンボリックリンク攻撃、およびハードリンク攻撃の防止。

### 課題4:

Cプログラマは、プログラム本来の中心的な機能の作成に加え、特定のセキュリティメカニズムを導入できなければならない。

#### 01.4.1 識別、認可、および認証

プログラマは認証を適切に実施し、攻撃者が認証を回避して、制限付きリソースへのアクセス権を得ることを阻止しなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 各種認証(2要素認証など)の相対的な強度と、これらの認証がユーザビリティに及ぼす影響について基本的な知識を得る。
2. リプレイ攻撃とスプーフィング攻撃を防止する。
3. パスワードやアカウント名はハードコーディングしない。
4. 適切なハッシュを使用してパスワードを保護する。
5. パスワードや鍵をプレーンテキストで保存しない。
6. 認証には、標準化されており、強度の高さが確認されているプロトコルとライブラリを使用する。

7. 保護するデータの価値に相応の認証を採用する。
8. 認証が必要なデータの価値に相応なセキュリティレベルが不明な場合は、いくつかのセキュアな認証方法を選択できるようにし、強度と心理的な受容度とのバランスが取れた認証の種類をデフォルトとして設定する。
9. アクセス権の決定では、特定のものを拒否するのではなく、特定のものを許可する方法に基づく。
10. リソースへのアクセス制限におけるアクセス権の役割、権限に関連したリスク、適切な承認の必要性を理解する。
11. 特定のリソースに対するユーザの承認を常に検証する。ユーザが直接変更可能なデータを使用して、承認の可否を決定しない。

#### スキル:

1. 代替名攻撃、代替パス/チャンネル攻撃、リプレイ攻撃、およびスプーフィング攻撃による、認証の回避の防止。

#### 01.4.2 プライバシーと暗号化

プログラマは、保管中や転送中のデータを保護し、適切な暗号化や復号を正しく実行しなければならない。プログラマは、信頼できない第三者への重要な情報の漏洩を防止しなければならない。プログラマは、ハッシュも含め、よく使われる暗号化方式の相対的な長所と短所を理解しておく必要がある。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. プライバシー保護における暗号化の役割を理解する。
2. 独自の暗号化方式を絶対に開発しない。熟練した暗号研究者による徹底的かつ公的な評価も行われた、定評ある暗号化方式のみを使用する。
3. 暗号化機能と、一意または予測が難しい識別情報(ユーザID、ファイル名、セッションIDなど)の生成において、ランダム性が重要な役割を果たすことを理解する。  
呼び出す関数により、統計的ランダム性ではなく暗号的ランダム性が実現することを検証する。
4. 暗号化操作のランダムデータを生成するときには、可能な限り多くのソースからデータを収集し、これらのデータを追加して1つの長い文字列を作成する。さらに暗号的にセキュアなハッシュアルゴリズムを用いてデータをハッシュ化し、最終的なランダム値を生成する。
5. 可能な場合は、定評ある標準インターフェースを使用して暗号ランダムデータを生成する(ただし、`C99 rand()`関数は予測可能な値を生成するため、使用してはならない)。
6. エントロピーが高いソースを使用する。ただし、エントロピーが低い場合でも処理可能であること。使用する技術により、ソースが拒否したり、ランダム性が必要なレベルよりも低い数値が生成される可能性がある。
7. 疑乱数生成機能には、外部者が予測できないシードを使用する。

プロセスIDやシステム時刻などの情報は多くの場合予測可能である。十分に広い範囲で乱数が生成されるようにする。これにより、ブルートフォース攻撃で解読される可能性も低減する。

8. パスワードまたは暗号鍵を保管しない。代わりに、パスワードまたは鍵のハッシュを保管する。これにより、比較対象として元の値を保管しておかなくても、指定された値を検証することができる。  
復元にパスワードまたは暗号鍵が必要な場合は、単独の媒体に鍵を保管する。特に重要な鍵の場合は、キーエスクロー手法または分割手法を使用する。
9. プロセスにより「コアダンプ」が出力され、ディスクへメモリイメージが書き込まれるのを防止する。
10. エラーメッセージやログ、そしてそのほかいずれの情報源にも、実装の詳細やパスワードなど重要な情報をフィードバックしない。
11. あいまいな手法でセキュリティを導入しない。ソフトウェアに組み込むセキュリティメカニズムはすべて、敵対者によりリバースエンジニアリングが可能であると仮定する。
12. デバッグシンボルを取り除き、ソフトウェアのバージョンを表示させないようにする。
13. 通信内容または受信データが、送信元もしくは通信経路上で改竄できないことを検証する。
14. クライアント（またはサーバー）のプログラムが、データを送信したり、正規プロセスでは実行されないアクションを実行したりするように変更または改悪される可能性があることを想定する。
15. 一般的な情報保護方式の限度を理解する。
16. 処理するデータのプライバシーを理解し、適切に保護する。
17. データを別のプロセスに渡す前に、ユーザが直接コントロールできない領域（一時フィールドなど）から重要なデータをすべて削除しておく。

#### スキル:

1. 不十分なランダム性、重要な情報のプレーンテキストでの保管、弱い暗号化レベル、可逆な一方方向性ハッシュ、必要な暗号化ステップの欠落、キー管理エラー、PRNGのエントロピー不足、予測可能性、およびランダム値の領域が小さいことに起因する、脆弱性の防止。

### 01.4.3 セキュアな設計

プログラマは、セキュアな設計を開発もしくは選択し、正しく実装できなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. プログラマは、セキュアな設計を作成するため、そして脆弱性の原因となる実装エラーを防ぐために、最も一般的な脆弱性と攻撃の背景にある基本概念や用語を理解しておく必要がある。
2. 可能な限り最も低い権限レベルで実行する。権限を持つタスクは早期に実行し、できる限り早い段階で権限を削除する。権限の削除の際は適切な手順に従う。可能であれば、特権コードとその他のコードをそれぞれ別個のコンポーネントに分け、これらのコンポーネントの間のアクセス権

限を慎重に検証する。信頼できないデータ送信元と対話するコンポーネントは、低い権限で実行できる設計を検討する。可能な権限の組み合わせをすべて洗い出し、複数の権限を組み合わせる必要以上に高い権限を取得することができないように検証する。

権限削除の処置が正常に実行されること、また失敗した場合でも適切に処理されることを検証する。

3. プログラムにより信頼できる内容と信頼できない内容の境界が不明確になった場合に生じる、信頼境界違反を回避する。この違反が発生する最も一般的な状況は、信頼されているデータと信頼できないデータ、またはデータとコントロールフロー情報が1つのデータ構造の中に混在可能な場合である。
4. セキュリティ関連イベント（ログイン、ログオフ、信用証明情報の変更など）をログに記録する。

#### **スキル:**

1. セキュアな設計の開発、選択、および実装。

## 課題5:

Cプログラマは、並行スレッド、プロセス、およびタスクに起因する問題をはじめとする、並行性の問題に対処できなければならない。

### 01.5.1 並行性

プログラマは、競合状態やデッドロックなどの不適切な動作の原因となる並行フローの順序での実行を防ぐ必要がある。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. マルチスレッドプログラムは、競合状態、デッドロック、二重ロック、ロック解除忘れ、優先度エラー(優先度の逆転、無限の追い越し(**Infinite Overtaking**))、およびリソース枯渇(**Starvation**)などを防ぐため、慎重に設計されなければならない。攻撃者がこれらの問題を引き起こしたり、または悪化させたりする可能性がある。
2. 競合状態を避けるため、共有データへのアクセスには原子操作を使用する。原子操作が使用できない場合は、共有データ(ファイルまたは共有メモリいずれの場合でも)へのアクセスをシリアライズするため、ロックを使用する。ロックや共有メモリなどの並行リソースは、制御下にあるプロセスやスレッドからのみアクセス可能であるようにする。
3. ある操作のセキュリティが特定の操作でチェックされ、その後の操作が実行される状況におけるTOCTOU(チェック時から使用時まで)の問題を防ぐため、独自のアクセスコントロールを設計せずに、オペレーティングシステムのアクセスチェックを使用し(**access()**は絶対に使用しない)、ファイル操作をファイル名ではなく、開いているファイル記述子に対して実行する(たとえば**chmod()**の代わりに**fchmod()**を使用する)。
4. コンパイラがスレッドセーフなコードを生成すると信頼してはならない。シグナルハンドラから非リエントラントライブラリ関数を呼び出さないこと。
5. データ転送中の接続中断など、非同期イベントが発生する可能性があることを見込む。

#### スキル:

1. シグナルハンドラ競合状態、TOCTOU(チェック時から使用時まで)、デッドロック、リソース消耗に起因する脆弱性の防止。

## 課題6:

Cプログラマは、組み込みデータ型と、これらの基本的な型をベースにした複合データ型の使い方を理解しておく必要がある。

### 01.6.1 NULL終端バイト文字列 (NTBS)

プログラマは、NULL終端バイト文字列を正しく使用し、よりセキュアな代替手段を把握しておく必要がある。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 有限長文字列をコピーするには、十分なスペースを割り当てる。
2. すべてのNTBSがNULLで終端しており、NULLバイトが誤った位置に書き込まれたり、後のコードにより上書きされたりする原因となるoff-by-oneエラーがないことを保証する。
3. 文字列を誤って切り捨てることのないようにする（特にコピー中）。
4. セキュアな文字列ライブラリ（CERTの文字列操作ライブラリまたはStrSafeなど）をできる限り使用する。
5. 入念に設計したパーサーを使用して文字列を検証する。標準の文字列ベースのフォーマット(XMLなど)の解析には、標準ライブラリを使用できる。標準ライブラリはカスタマイズして作成したパーサーよりも綿密に検査できることが多い。
6. NULL終端バイト文字列を使用する際に、一般的なエラーが発生しないようにする。
  - a. オブジェクトの型が必ずしも同一でない場合は、ポインタを使用したオブジェクトのコピーは避ける。
  - b. バッファサイズをバイト単位で受け入れる関数に対し、バッファ長を文字単位で指定しない。また、この逆の指定も行わない。
  - c. 可能な限り、混乱を避けるため1つの規定に基づく関数のみを使用する。
  - d. ユーザ入力によってループの終了条件に影響を与えないようにする。
  - e. 文字列の比較にポインタ値を使用しない。ただし、文字列に「アトム」パターンが使用されている場合を除く。
  - f. 標準C文字列ライブラリ関数を使用するときには、常にNULL終端文字も考慮した、十分なスペースを割り当てる。

#### スキル:

1. バッファオーバーフロー、NULL終端エラー、および文字列切り捨てエラーに起因する脆弱性の防止。

### 01.6.2 整数

プログラマは、整数値を正しく管理するため、符号の有無、オーバーフロー、切り捨て、および取

り得る範囲（境界）に関連するエラーを検出して回避しなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 整数操作の結果が、無限範囲の整数を使用した場合の結果と異なる場合は、その操作の使用を避ける。特に、整数オーバーフロー、切り捨て、符号エラーに起因する予期しない値を排除する。
  - a. 数値の精度が特定のビット幅に一致している必要があるコードでは、明示的な幅を持つ整数型(`uint32_t`、`int16_t`、`DWORD`など)を使用する。
  - b. オブジェクトのサイズを表すときには`size_t`型または`rsize_t`型を使用する。
  - c. 負数となってはならない値(長さ、ほとんどの場合の配列インデックス、オフセットなど)には、符号なし型を使用する。
  - d. `char`型に符号が付いているかどうかを仮定しない。
  - e. 変換によってデータの欠落やデータの誤変換が生じないことを確認する。
  - f. 可能な限りセキュアな整数ライブラリ(`IntSafe`や`SafeInt`など)を使用する。
  - g. 演算操作によってオーバーフローが発生しないこと、またはプログラムがこのようなオーバーフローを検出して正しく処理することを検証する。

#### スキル:

1. 整数値のオーバーフロー、整数値の符号エラー、整数値の切り捨て、および整数値範囲エラーに起因する脆弱性の防止。

#### 01.6.3 その他のデータ型

プログラマは、配列、浮動小数点数、ポインタ、オブジェクトなどのその他のデータ型を正しく管理できなければならない。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 配列
  - a. `sizeof`演算子を使用して配列サイズを判断する際には、注意してこの演算子を使用する。
  - b. 配列インデックスが有効な範囲内であることを保証する。
  - c. すべてのソースファイルで一貫した配列表記を使用する。
  - d. 可変長配列のサイズ引数が有効範囲内であることを確認する。
2. 浮動小数点
  - a. 浮動小数点数値を比較するときには、粒度を考慮する。
  - b. 浮動小数点変数をループカウンタとして使用しない。
  - c. 数学関数での定義域エラーを防ぐ。
3. ポインタ
  - a. ポインタへの整数値のキャストを回避する。

- b. 同じメモリオブジェクトの一部を指し示していないポインタに対してポインタ演算を使用しない。
  - c. ユーザ入力によって関数ポインタを指定できないようにする。
4. 構造体
    - a. データ構造体で使用されるすべてのオフセットを検証する。
    - b. 構造体間でバイト単位の比較を実行しない。
  5. 構造体のサイズと、その構造体のメンバーのサイズの合計が等しいと仮定しない。

**スキル:**

1. 索引エラーおよびバッファオーバーフローに起因する脆弱性の防止。

## 課題7:

Cプログラマは、すべてのエラー状態を正しく処理できなければならない。

### 01.7.1 戻り値

プログラマは、各関数呼び出しから戻される可能性があるすべての戻り値(エラーや異常な状態を含む)に対処できなければならない。

#### セキュアコーディングに関する規則の理解と適用:

1. エラー状態を戻す可能性があるすべての関数の戻り値を検査する。ただし、関数が失敗する恐れのある条件が今後絶対に発生しないこと、または関数の失敗により、プログラムの正常動作、およびセキュリティレベルなどに異常を引き起こす可能性がないことを実証できる場合を除く。  
`close()`など、実際に失敗する可能性がないと考えられる関数でも、特定の状況下で失敗することがあるので注意する。
2. 関数の失敗が回復不能なエラーの場合、少なくとも特定のバージョンで関数をラップし、戻り値を検査してエラー発生時には強制終了するようにする。
3. 常に`malloc()`の戻り値を検査し、`nothrow`構文を利用した`new`演算子の呼び出しも確認する。  
発生する可能性があるすべての戻り値を検証しておく。

#### スキル:

1. 検査されていない戻り値に起因する脆弱性の防止。
2. 権限昇格、シンボリックリンク攻撃、およびハードリンク攻撃の防止。

## 課題8:

Cプログラマは、正確にコードを記述し、有効なコーディングスタイルを使用しなければならない(これはあらゆる状況に適用される)。

### 01.8.1 コーディングスタイル

プログラマは、脆弱性が誤って生じる可能性を最小限に抑え、既存の脆弱性とセキュリティ上の弱点を検出できる可能性を高める方法で、コードを理解可能かつ維持可能であるようにする規則に従う必要がある。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 一貫性のあるコーディングスタイルとフォーマットを使用する。
2. 論理的に完全である状態を目指す。
3. コメントは読みやすく一貫性のある方法で記述する。
4. 識別子については以下の規則を適用する。
  - a. コードの開発と見直しの段階において識別子のスペルの誤読が原因で発生するエラーを除去するため、視覚的に明確に区別できる識別子を使用する。
  - b. 1つもしくはいくつかの、類似する文字のみが異なる識別子を複数定義しない。
  - c. 長い識別子の場合は、すぐに見分けることができるように、識別子の先頭部分を固有にする。
  - d. 変数、関数、およびその他の名前付きオブジェクトの名前を指定するときに、一貫性のある命名規則を採用する。
  - e. 長くなり過ぎないように気を付けつつ、説明的な名前を指定するようにする。
  - f. 短い名前を指定するための方針を定める。(例: "int"は常に"i"で表し、"符号なしint"は常に"u"で表すなど)。
  - g. 1つのスコープがもう1つのスコープに包含されている場合、この2つのスコープ内で同名の変数を使用しない。
5. 演算の前後を括弧で囲む。
6. セキュリティの見地から危険であると認識されている関数を使用しない。誤った使い方を招く恐れのある関数を使用しない。危険性のある関数のうち問題のないバージョンのものを使用するか、またはセキュアな設計方針に従って関数を全体的に記述し直す。危険性のある関数の使用が必要な場合は、関数の制約を文書化し、このような関数を使用するコードに細心の注意を払い、セキュリティ上の弱点の悪用を助長しない方法でこれらの関数が使用されるようにする。
7. コードレビューにおいて、未使用コードを洗い出し削除する。どのようなユーザ入力においても使用されることのないコードは、上記に挙げた未使用コードに該当するが、「不可能な(はずの)」条件において使用されるコードは未使用コードではない。この種のコードは、堅牢性の点で重要である。
8. 使用されていないデバッグコードをソフトウェアから削除する。代わりに、条件付きコンパイル

構造の中に、追加のデバッグコードを格納し、明示的にデバッグレベルフラグを使用し、デバッグコードをテストする。

9. 定数については以下の規則を適用する。
  - a. 不変値を`rvalue`または変更できない`lvalue`として宣言する。
  - b. オブジェクトの状態を変更しないオブジェクトメソッドに`const`修飾子を付ける。
  - c. データメンバをプライベートとして宣言し、プライベート属性へ参照戻しを行うときに`const`を使用する。
  - d. `const`修飾をキャストで除去しない。
10. プリプロセッサについては以下の規則を適用する。
  - a. プリプロセッサは慎重に扱う。
  - b. マクロよりもインライン関数を優先して使用する。
  - c. マクロ内では変数名を括弧で囲む。また、マクロ拡張は常に括弧で囲まなければならない。
11. 処理系定義の動作を使用しないようにする。および/または、想定事項を文書化する。
12. 精度に関する要件が判明している場合は、固定精度の標準型(`int32_t`など)を使用する。
13. システムに依存する仮定事項を可能な限り最小限に抑えるコードを作成する。システム依存操作が必要な場合は、システム依存モジュールでこのような操作を分離する。
14. コンパイラのセキュリティメカニズム（バッファオーバーフローの保護機能などを含む）の使用を評価する。

#### スキル:

1. 新たな脆弱性が発生する可能性を抑え、既存の脆弱性を検出できる可能性を高め、読みやすく理解しやすいコーディングスタイルの定義と適用。

#### 01.8.2 正しいコーディング

プログラマは常に正しいコードを記述しなければならない。誤ったコードがプログラムの通常の実行状況に影響を与えない場合でも、関連するリスクがあるため、同様である。

#### 理解し適用すべきセキュアコーディングに関する規則:

1. 文字の欠如や誤追加によるエラーを防ぐ。必要な文字が欠如しているコードや、誤って文字が追加されているコードは、クリーンコンパイルが成功したとしても予期しない動作を生じる可能性がある。たとえば、条件式の中で等号“=”が誤追加または欠如され、誤って代入演算子と比較演算子を混用することがないようにする。
2. 型のサイズを判別する目的でポインタに対して`sizeof`を使用しない。
3. `sizeof`演算子のオペランドに副作用があってはならない。
4. 論理AND演算子または論理OR演算子の2番目のオペランドには副作用があってはならない。
5. `const`修飾をキャストで除去しない。

6. シーケンスポイント（副作用完了点）間の評価順序に依存してはならない。また、未初期化変数を参照してはならない。
7. 生存期間を超えたオブジェクトを参照しない。
8. フェッチ操作と格納操作は、すべて既存のオブジェクトの境界内で実施される必要がある。配列要素のフェッチ操作と格納操作はほとんど検査されないため、この規則はこれらの操作において特に重要である。
9. すべての変数が参照前に初期化されていることを確認する。
10. 数値を格納する変数が、数値の精度に対応していることを明確にできない限り、記述するコードにおいて数値の精度を仮定しないようにする。
11. 高い警告レベルを指定してクリーンコンパイルを実行する。

**スキル:**

1. 初期化エラーに起因する脆弱性の防止。

## 付録B:参考文献

以下のリストは、プログラマーやアプリケーション開発者、セキュリティ従事者にとって、セキュアプログラミングについてさらなる学習の一助となる資料の例です。アセスメントに役立つ資料も含まれます。これは、試験対策の関する完全なリストでも GIAC からの推薦リストでもありません。単に、学習される方の興味を引くきっかけになり得るものとして紹介しています。

### ソフトウェアセキュリティに関する参考書

#### **19 Deadly Sins of Software Security**

Michael Howard, David LeBlanc, John Viega

#### **Building Secure Software: How to Avoid Security Problems the Right Way**

John Viega, Gary McGraw

#### **Exploiting Software: How to Break Code**

Gary McGraw, Greg Hoglund

#### **Foundations of Security: What Every Programmer Needs to Know**

Neil Daswani, Christoph Kern, Anita Kesavan

#### **Introduction to Computer Security**

Matt Bishop

#### **J2EE & Java: Developing Secure Web Applications with Java Technology (Hacking Exposed)**

Art Taylor, Brian Buege, Randy Layman

#### **Secure Coding in C and C++**

Robert Seacord

#### **Secure Coding: Principles and Practices**

Ken Van Wyk, Mark Graff

#### **Hacking Exposed: Web Applications**

Scambray, Shema, Sima

#### **Secure Programming Cookbook for C and C++**

John Viega, Matt Messier

#### **Security and Usability**

Simson Garfinkel, Lori Faith Cranor

#### **Software Security: Building Security In**

Gary McGraw

#### **The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities**

Mark Dowd, John McDonald, Justin Schuh

**The Security Development Lifecycle**

Michael Howard, Steve Lipner

**Web Security, Privacy & Commerce, Second**

Simson Garfinkel, Gene Spafford

**Writing Secure Code, Second Edition**

Michael Howard, David C. LeBlanc

[ソフトウェアセキュリティに関する Web サイトや Podcast](#)

情報処理推進機構 (IPA) 「安全なウェブサイトの作り方」

<http://www.ipa.go.jp/security/vuln/websecurity.html>

**OWASP - Open Web Application Security Project**

[http://www.owasp.org/index.php/Main\\_Page](http://www.owasp.org/index.php/Main_Page)

**Microsoft Corporation - Security Developer Center**

<http://msdn2.microsoft.com/en-us/security/aa570401.aspx>

**MITRE - Common Weakness Enumeration (CWE)**

<http://cwe.mitre.org/>

**CERT - Secure Coding Initiative**

<http://www.cert.org/secure-coding/>