ソースコード検査に耐えるコードとは？
～Webアプリ編 2016年度版～

エレクトロニック・サービス・イニシアチブ
http://www.es-i.jp/

コンテンツ

- セキュリティとは？
- セキュリティ標準とベストプラクティス
- セキュアなコードとは？

セキュリティとは？

基本的な要素を簡単に解説します

セキュリティ対策　＝　セキュリティマネジメント

ISO27000の「リスク対応」（Risk Treatment）の定義

- *リスクを変化させる為のプロセス*
  - *リスクを発生させる活動を開始しない、または継続しないことを決断することによりリスクを回避する*
  - *機会を獲得する為にリスクを選択または増加させる*
  - *リスクの原因を排除する*
  - *発生の頻度を変える*
  - *結果を変える*
  - *他の組織（契約企業やリスクの資金的処理を含む 訳注：保険など）とリスクを共有し*
  - *見聞のある選択によりリスクを抑える*
- *否定的な結果をもたらす事象のリスク対応は、”**リスク緩和策**”、”**リスク排除策**”、”**リスク防止策**”、”**リスク削減策**”などと呼ばれる事がある*

リスク（脆弱性）の排除や削減はセキュリティ対策の一部でしかない

✓ 国際ITセキュリティ標準ISO 27000の定義

リスク緩和策はセキュリティ対策

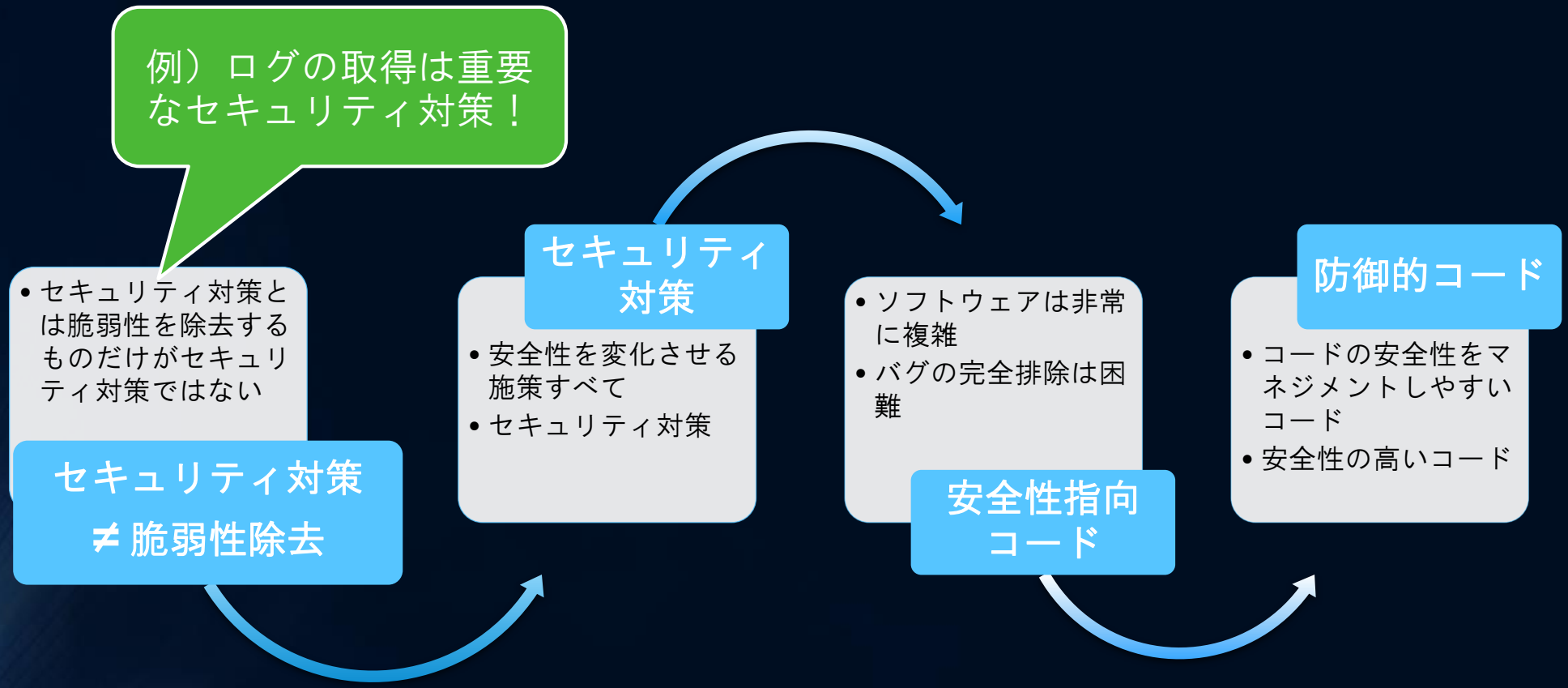
- 否定的な結果をもたらす事象のリスク対応は、
  - **〝リスク緩和策〟**
  - **〝リスク排除策〟**
  - **〝リスク防止策〟**
  - **〝リスク削減策〟**
- などと呼ばれる事がある

POINT！ 緩和策をセキュリティではない
と勘違いしない

POINT！ 排除策、防止策、削減策は
セキュリティ対策の一部

✓ 国際ITセキュリティ標準ISO27000の定義

セキュリティ定義とセキュアコード

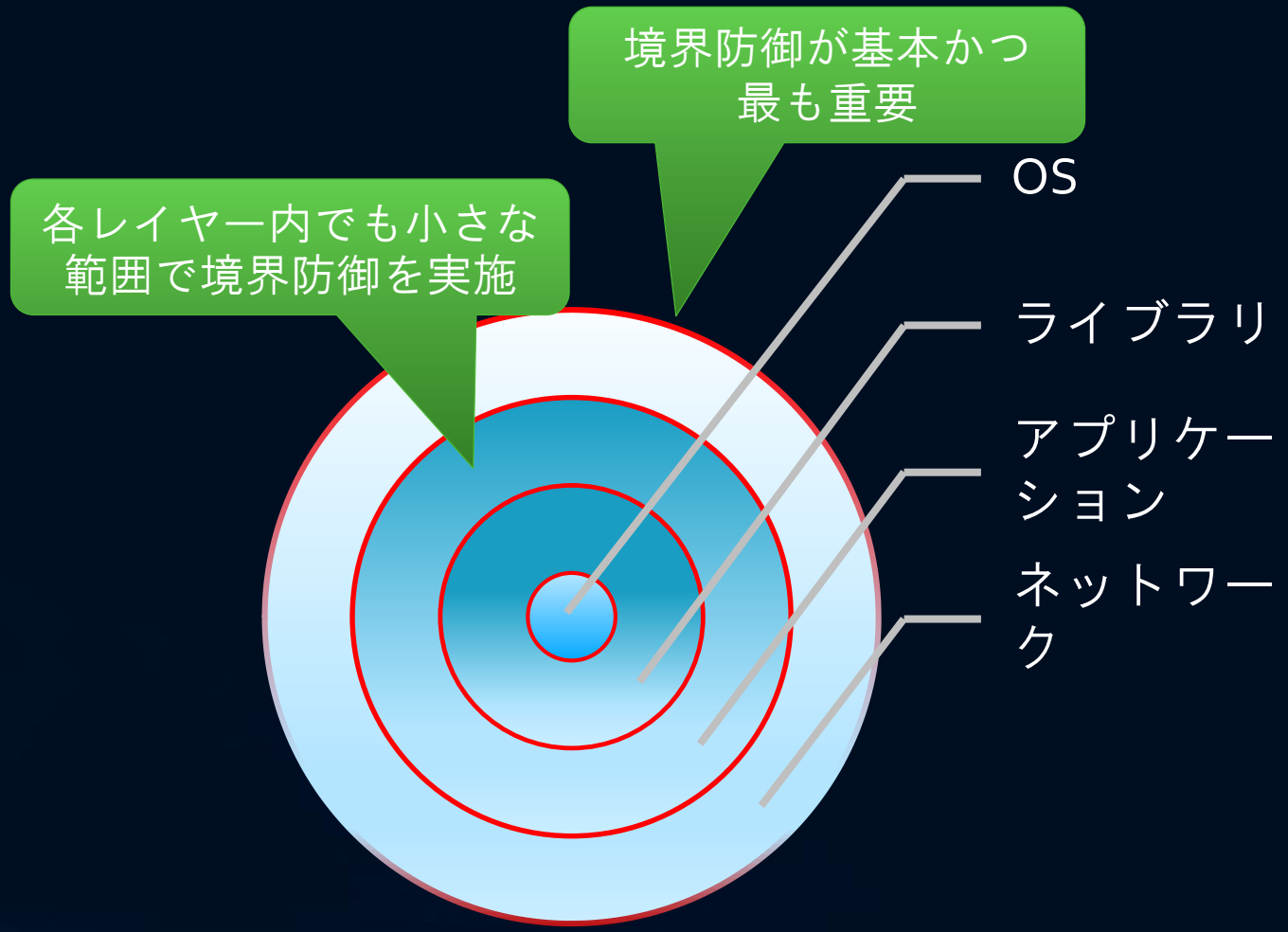


✓ 防御的プログラミングは「プログラミング原則」のひとつ

セキュリティの基本概念

- 境界防御
  - 信頼境界線で防御
- 縦深防御（多層防御）
  - 信頼境界線内で防御

✓ アプリケーション内の
縦深防御（多層防御）が重要

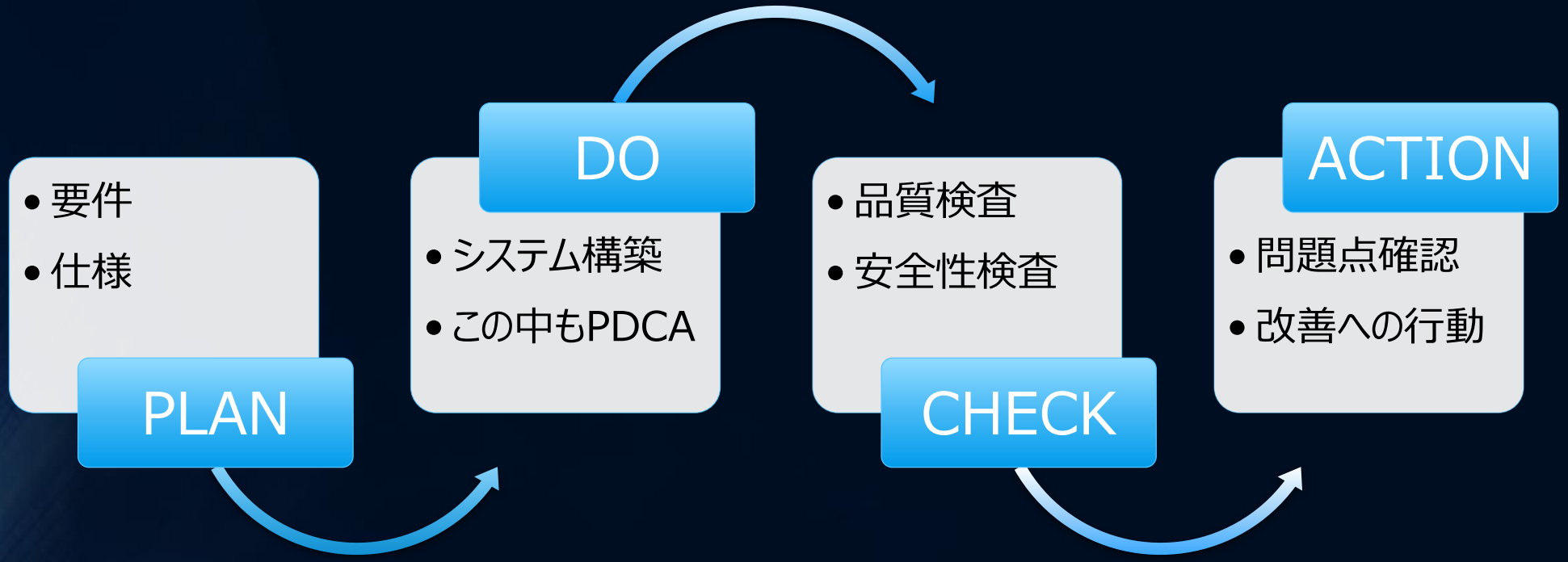


セキュリティの6要素
＝　セキュアなアプリケーションの要素

- Confidentiality（機密性）
- Integrity（完全性）
- Availability（可用性）
- Reliability（信頼性）
- Authenticity（真正性）
- Non-Repudiation（否認防止）
  Accountability（追跡性）

✓ 国際ITセキュリティ標準ISO 27000の定義

セキュアな開発はPDCAサイクルに対応



✓ PDCAマネジメントは国際標準ISO規格の根幹

セキュリティ標準とベストプラクティス

開発者が実践すべき標準とプラクティス

セキュリティ標準規格

- **ISO27000**
  - 国際標準のITセキュリティ標準規格
  - JISとしても定義
  - IT開発者必修
  - http://www.iso.org/iso/home.html
- **PCI DSS**
  - カード会社が中心となって策定した規格
  - クレジットカードなどの取り扱いに必須
  - https://ja.pcisecuritystandards.org/minisite/en/

Web開発者必修ベストプラクティス

- **CERT Top 10　Secure Coding Practices**
  - 米カーネギーメロン大学のCERTが公開しているセキュアコーディング標準トップ10
  - CERTは1988年に設置され、ソフトウェアセキュリティの草分け的な組織として現在に至る
  - https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices
- **OWASP Secure Coding Practices – Quick Reference Guide**
  - OWASP（Open Web Application Security Project）：PCS DSS標準でも言及されているセキュリティ推進団体
  - 多数公開されているガイドの対策をまとめた文書。チェックリスト形式でセキュアコーディングの実施状態を確認できる。
  - https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

CERT Top 10 Secure Coding Practices

- 1. 入力をバリデーションする
- 2. コンパイラの警告に用心する
- 3. セキュリティポリシーの為に構成／設計する
- 4. 簡易にする
- 5. デフォルトで拒否する
- 6. 最小権限の原則を支持する
- 7. 他のシステムに送信するデータを無害化する
- 8. 縦深防御を実践する
- 9. 効果的な品質保証テクニックを利用する
- 10. セキュアコーディング標準を採用する

http://blog.ohgaki.net/cert-top-10-secure-coding-standard

OWASP Secure Coding Practices – Quick Reference Guide

- 入力バリデーション
- 出力エンコーディング
- 認証とパスワード管理
- セッション管理
- アクセス制御
- 暗号の取り扱い
- エラー処理とログ
- データ保護
- 通信セキュリティ
- システム設定
- データベース管理
- ファイル管理
- メモリ管理
- 一般的コーディング実践

http://blog.ohgaki.net/owasp-secure-coding-practices-quick-reference-guide

Web開発者必修ガイドライン

- **CWE/SANS TOP 25**
  - 最も危険なアプリケーション脆弱性トップ25とその対策
  - CWE（Common Weakness Enumeration）：米国MITRE社の脆弱性カタログ
  - SANS：セキュリティ研究・教育を行う米の組織
  - https://www.sans.org/top25-software-errors/
- **OWASP TOP 10**
  - Webアプリケーションに特化した最も危険な脆弱性トップ10
  - https://www.owasp.org/index.php/Main_Page

開発者必修プログラミング原則

- **Defensive Programming（防御的プログラミング）**
  - Secure　Programmingとも呼ばれる
  - バグを最小限に留めるためのプログラミングプラクティス
    - Intelligent source code reuse
    - Legacy problems
    - **Secure input and output handling**
    - Canonicalization
    - Low tolerance against "potential" bugs
    - **All data is tainted until proven otherwise**
    - All code is insecure until proven otherwise
    - Design by contract
    - Assertions
    - Prefer exceptions to return codes
    - その他

安全な入出力の管理
これを行うだけでもWebアプリのセキュリティ問題の大多数を解決可能

✓ 安全な入出力管理以外の原則は入出力管理に関連
✓ またはそれらの緩和策と機能する

開発者必修CWE/SANS TOP25の「怪物的セキュリティ対策」

- CWE/SANS TOP25では個別の脆弱性対策以外に一般的対策として「怪物的セキュリティ対策」（Monster Mitigations）を定義
- 怪物的セキュリティ対策の第一位と第二位

一位：確実な入力の管理

- 入力バリデーションをシステム全体で行う

二位：確実な出力の管理

- 安全な出力をデフォルトで利用
- エスケープ、安全なAPI、バリデーション

安全な入出力の管理
これを行うだけでもWebアプリのセキュリティ問題の大多数を解決可能

✓ プログラミング原則 – 防御的プログラミング
✓ ISO27000 – 同様の要求

セキュアなコードとは？

アプリケーション開発者が知るべきセキュアなコードの特徴

ソフトウェアは非常に複雑

- 簡単なコードでも背後で動作している多数の関数
- しかもこれらは頻繁に更新
  - 機能追加・リファクタリング・バグフィックス
  - アプリ自身は更新しなくても、ライブラリやプラットフォームのバージョンが更新
  - 攻撃者はたった一つの弱点を見つければ勝ち

✓防御的プログラミングが有効

IISのコールグラフ

https://twittech.wordpress.com/2013/05/06/building-kernel-modules-with-autotools/

防御的プログラミングの概念とツール
– セキュアプログラミング

- 防御的プログラミングの要素
  - Intelligent source code reuse
  - Legacy problems
  - Secure input and output handling
  - Canonicalization
  - Low tolerance against "potential" bugs
  - All data is tainted until proven otherwise
  - All code is insecure until proven otherwise
  - Design by contract
  - Assertions
  - Prefer exceptions to return codes
  - その他

✓ 信頼できる、と保障できない物（入出力データ、ライブラリ、コード）は信頼しない

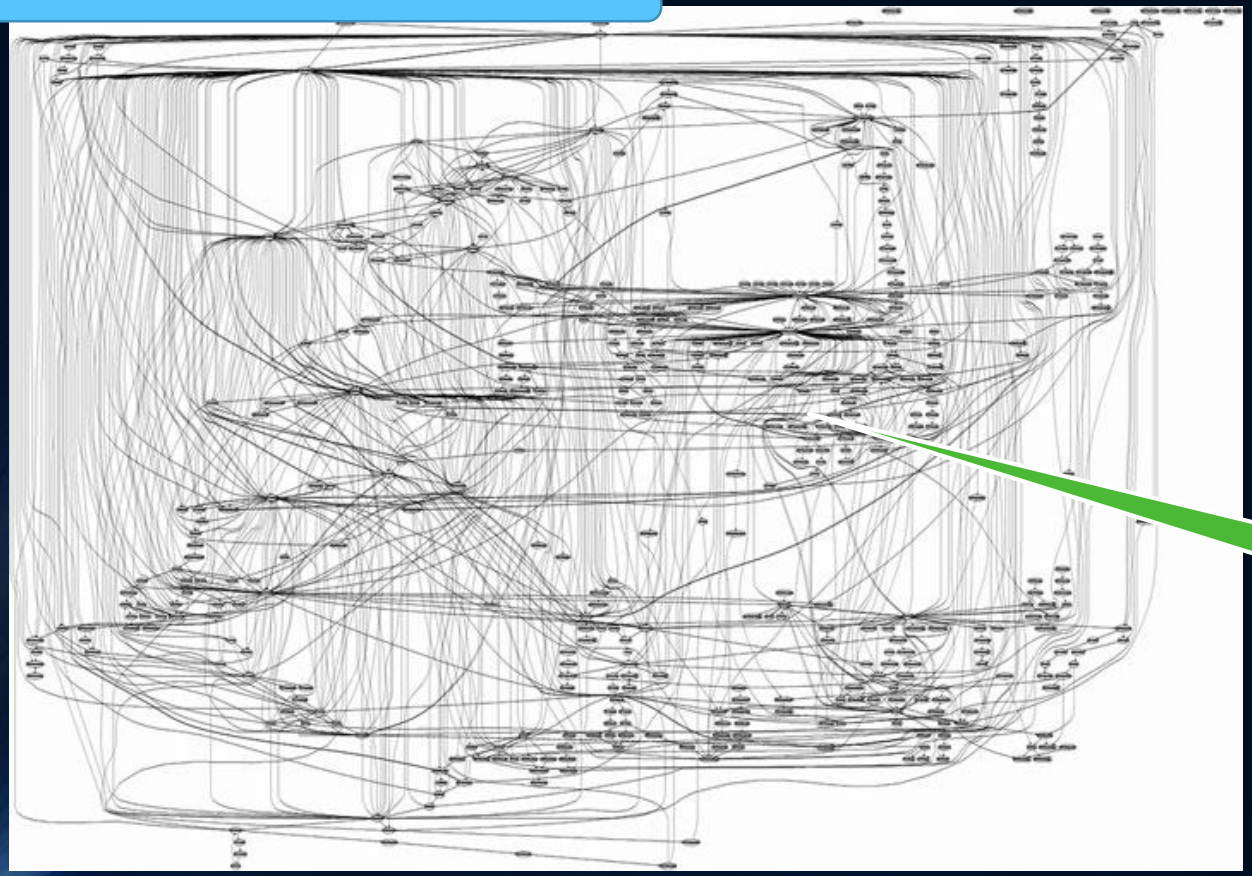
- セキュリティ標準・ガイドラインをツールとして利用
  - ISO27000
  - CWE/SAN TOP 25
  - CERT Secure Coding
  - OWASP TOP 25
  - OWASP Guide
  - OWASP Chart Sheet
- マニュアル・リファレンス
  - プロトコルの仕様
  - 外部システムの入出力仕様
  - ライブラリの仕様
  - 言語・関数の仕様

安全な入力・出力処理には正確な仕様の理解が欠かせない

✓ 防御的プログラミングには仕様の理解が不可欠

一般的なプログラムの構造

IISのコールグラフ



Webサーバーの基本動作は単純

リクエストを受信

何らかの処理

レスポンスを送信

基本動作は単純でも内部構造は複雑

防御のポイント

- **入力処理**
  - バリデーション（ホワイトリストが基本）
- **ロジック処理**
  - ベストプラクティス
    - OWASP・CWE・ISO・その他のガイドライン・標準を利用
- **出力処理**
  - エスケープ、安全なAPI、バリデーション（ホワイトリストが基本）

まずアプリケーション全体を防御

- **入力処理**
  - バリデーション（ホワイトリスト優先）
- **出力処理**
  - エスケープ、安全なAPI、バリデーション（ホワイトリスト優先）

✓ 境界防御 – セキュリティの基本原則

✓ 全ての入力をバリデーション
※ ISO27000の要求事項

入力処理でバリデーションし無用なリスクを排除

防御対象アプリケーション境界での保護が第一

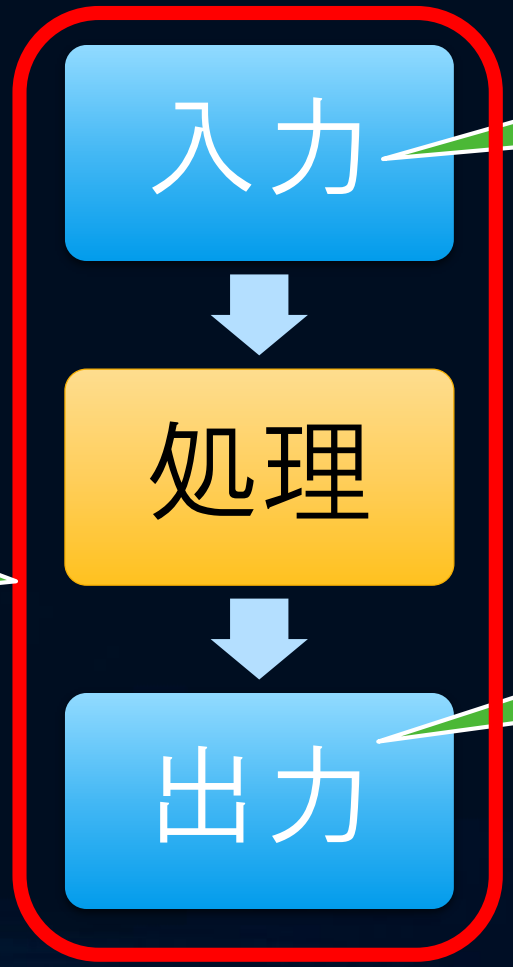
出力処理の安全性を確実に担保

✓ デフォルトでエスケープ
✓ 例）HTML出力

モジュール・関数レベルも同じ

- **入力処理**
  - バリデーション（ホワイトリスト優先）
- **出力処理**
  - エスケープ、安全なAPI、バリデーション（ホワイトリスト優先）

アプリ・モジュール・関数の境界を防御



入力処理でバリデーション

- ✓ 全ての入力をバリデーション
- ✓ Assertや契約プログラミングが有効

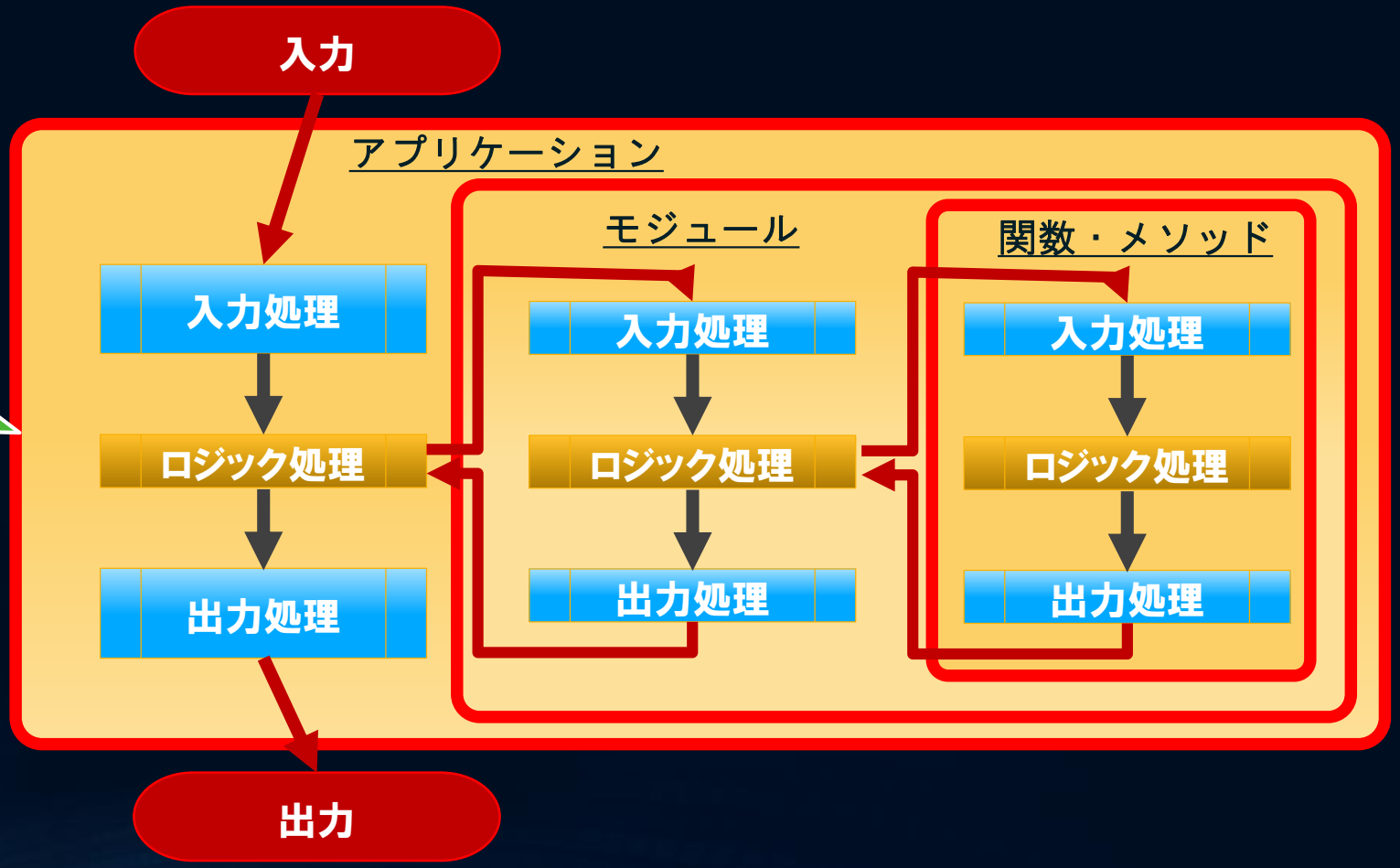
出力処理を既定として安全性を担保

- ✓ 異常な出力を防止（無効な値など）
- ✓ Assertや契約プログラミングが有効

- ✓ モジュール・関数レベルで見ると境界防御だが、アプリレベルと見ると**縦深防御（多層防御）**
- ✓ 縦深防御もセキュリティの基本原則

セキュアなソフトウェアアーキテクチャ

- 基本アーキテクチャ　– アプリケーション、モジュール、関数・メソッド、粒度に関わらず共通



各レイヤーの
境界防御
が防御の基本

✓ アプリケーションレベルの境界防御が特に重要

セキュアなコード
＝　セキュリティの基本要素を守るコード

- 入力・出力制御
- セッション管理
- 認証・認可
- 暗号・ハッシュ
- フェイルセーフ
- 認証ログ、操作ログ、エラーログ
- テスト・品質検査
- コードレビュー

ベストプラクティスを最大限に利用

✓ フレームワークやライブラリが必ずしもベストプラクティスを実装しているとは限らない点に注意

セキュアなコード
＝　コードの安全性がひと目で分かる

- PHPの例

パラメータが設定されているか
のみをチェック？！

NG

```
$id = isset($_GET['id']) ? $_GET['id'] : null;
```

```
$id = validate($_GET['id'], 'int', 0, PHP_INT_MAX); // invalidはエラーで停止
```

入力バリデーションはセキュリティ対策であり、
入力パラメータは可能な限り厳格にチェック
入力バリデーションエラーはプログラムの実行を停止

セキュアなコード
＝　コードの安全性がひと目で分かる

- PHP+HTMLの例

入力バリデーションで「入力」の安全性は担保
そのまま出力して安全？！

NG

```
<div id="<?php echo $integer_id ?>" > .....</div>
```

```
<div id="<?php echo hemlspecialchars($integer_id, ENT_QUOTES,
'UTF-8') ?>" > .....</div>
```

入力バリデーションは「入力」の「境界防御」
防御的プログラミングでは出力を行う場合にも「出力」
の「境界防御」を行う（**出力対策は入力対策と独立**）

Webアプリケーションの世界はテキストが基本

- 安全なテキスト処理がセキュリティ対策の基本
  - **入力・出力の仕様**を正確に理解
    - HTML、CSS、JSON、SQL、LDAP、XML、XPATH、SMTP、HTTP他
    - OSコマンド、C言語文字列、PHP文字列、JavaScript文字列他
  - **一文字**でも意味がある文字を持つ場合、インジェクション攻撃が可能と仮定
    - 実際、一文字でも特別な文字があるとインジェクション攻撃に脆弱に
      - ヌル文字インジェクション、改行インジェクション（SMTP、HTTP）
  - テキスト処理をする場合、まず特殊文字があるか調べる
  - 一文字でもある場合、エスケープ方法・関数があるか調べる
    - エスケープ方法が定義されていても、エスケープAPIが無い場合も多い
  - 安全なAPIがあれば利用する
    - 安全と言われるAPIでも制限はある。例）プリペアードクエリは識別子、SQL語句のパラメータ化は対応していない。変数も埋め込める。
  - バリデーションは“**ホワイトリスト方式**”が基本中の基本
    - バリデーションエラーになる無効な入力を受け付ける意味は通常ないので拒否する

正確・安全なテキスト処理を知り、実践する

↓

安全なコードへの最短距離

✓ APIさえ使っていれば大丈夫、は遠回りかつ危険

セキュリティと速度・コスト

- セキュリティ対策の多くはトレードオフ関係

境界防御、縦深防御（多層防御）、
防御的プログラミング

実行速度増加、実装コスト増加

防御的プログラミングを行わないコスト

- 防御的プログラミングを行わないメリット・デメリット

コード記述量が減る、一見システムが正常（安全）に動作しているように見える（構築期間短縮）、システムの動作が軽快

隠れたバグの増加、セキュリティに影響するバグの増加、構築期間が短く見えてもバグ修正のコストが増大

✓ スーパープログラマが全てのコードを一人で書けばバグフリーも可能だが、現実の開発ではほぼ不可能

CSRF - Cross Site Request Forgery

CSRFは真正性の問題

- 真正性はISO27000でセキュリティの要素として定義されている
- 利用者、プロセス、システム、情報などが本物であることを確実にするということ。

Webアプリケーションはリクエストの詐称が容易

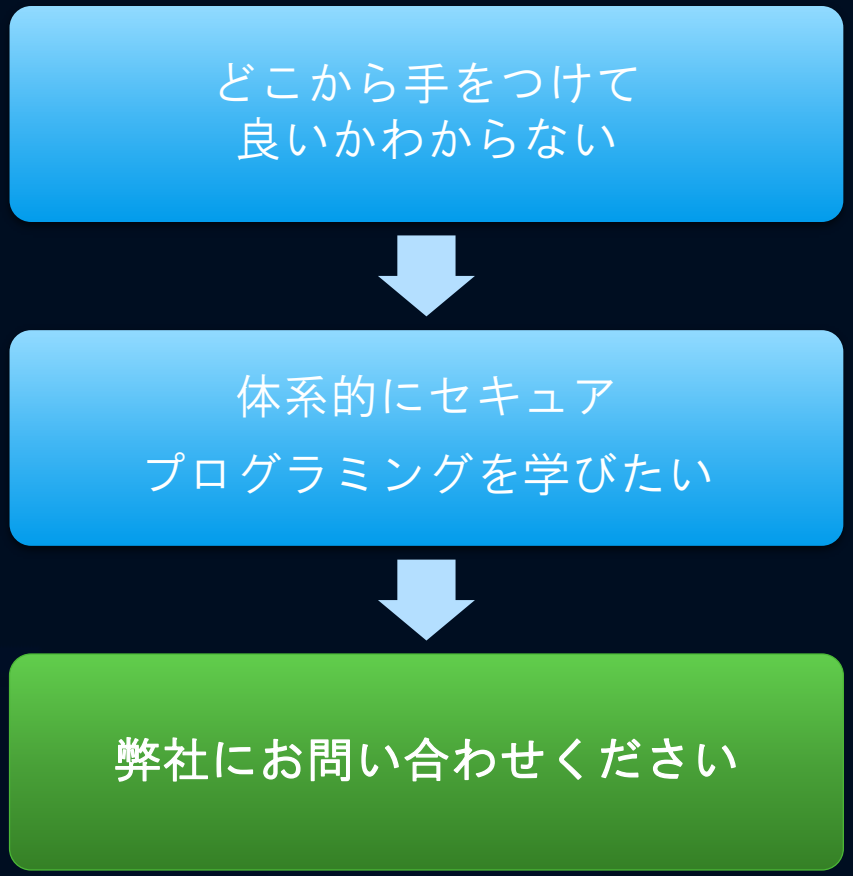
- 罠ページを用意して認証済みのユーザー権限で不正なリクエストを送信
- Webアプリケーションでは広くみられるセキュリティ問題

更新系のリクエスト全てにCSRF対策が必要

- リクエストにランダムかつ一意なトークンを付与し、サーバーでバリデーション
- CSRF対策は入力バリデーションの一種

セキュア（防御的）プログラミングの学習

- セキュリティ標準・ガイドライン
  - ISO27000
  - CERT Secure Coding Standard
  - CWE/SAN TOP 25
  - OWASP TOP 10
  - OWASP Secure Coding Practices
  - OWASP Guide
  - OWASP Chart Sheet
- マニュアル・リファレンス
  - プロトコルの仕様
  - 外部システムの入出力仕様
  - ライブラリの仕様
  - 言語・関数の仕様
- メソドロジー
  - 契約プログラミング（契約による設計）
  - OpenSAMM



まとめ

セキュリティ問題の8割、9割は
入力と出力のセキュリティ対策で対応可能

入力

- ホワイトリスト型でバリデーション
- 全ての入力値が対象、要素数などもチェック

入力バリデーションは
#1のセキュリティ対策
（SANS, MITRE, CERT,
OWASP, ISO）

出力

- 入力対策とは独立した対策。出力を徹底的に無害化
- エスケープ、エスケープが必要ないAPI、バリデーション処理

処理

- ベストプラクティスを採用。特に認証、認可、その他セキュリティに関連する処理

安全な入力処理の原則

入力バリデーション
エラーと入力ミスは
異なるモノである点
を理解していない開
発者も多い！

バリデーション

- **全ての入力データは厳格にバリデーションする**
  - バリデーションはホワイトリスト方式
  - 文字データ：文字エンコーディング、利用文字、形式、長さ
  - 数値データ：整数、浮動小数、最小値、最大値
  - 集合データ：集合に合致する値　例）都道府県、性別、学年
  - 入力値：入力パラメータの数、必須入力

✓ **入力バリデーションは最も重要なセキュリティ対策**
✓ **バリデーションエラーは処理を停止**

安全な出力処理の三原則

「セキュアなAPIを利用する」は不完全な対策であることがある点に注意！

1．エスケープ

- 出力先のエスケープ方式理解し確実にエスケープする
- 出力先が用意しているエスケープ関数を利用する
- エスケープ関数が無い場合もある

2．API利用

- エスケープが必要なくなるAPIが利用可能な場合は利用する
- セキュアなAPIが無い場合が多い

3．バリデーション

- エスケープもAPIも利用できない場合、バリデーションを行う
- バリデーションできない場合、サニタイズを行う

✓ **出力処理は入力処理とは“独立”である**
✓ **正しい文字エンコーディングであることを保証する**

セキュリティ問題の8~9割に
入力処理と出力処理で対応可能

入力バリデーションをセキュリティ対策でない、
と考えている開発者は少なくないので要注意

- **入力バリデーションはソフトウェア開発で最も重要なセキュリティ対策**
- 入力バリデーション無しで喜ぶのは攻撃者とブラックボックステストをするセキュリティ専門家
- 入力バリデーションはISO27000（ISMS）、PCI DSSで要求している事項。**入力バリデーションを実施しない場合、仕様書に記載がなくても開発会社は損害賠償責任**を問われる可能性がある

入力対策と出力対策が独立した対策であること
を理解していない開発者は多いので要注意

- 入力時に出力時のコンテクストは不明。したがって、**入出力処理は独立した対策**
- 入力・出力対策（境界防御）はITセキュリティ以外にも通用するセキュリティ対策の基本中の基本

✓ **入力と出力の制御は第一位、第二位のセキュリティ対策**
✓ **第一位、第二位のセキュリティ対策無しの対策は有り得ない**

ベストプラクティス

CWE（Common Weakness Enumeration - 共通脆弱性タイプ一覧）にはソフトウェアによく見られる脆弱性とその対策が記載されている

- https://www.ipa.go.jp/security/vuln/CWE.html
  https://cwe.mitre.org/
- CWEが利用する言語はJava、C#が多いが他の言語でも参考になる
- CWEに記載されている対策が最善の対策とは限らない
- CWEに記載されていない脆弱性もある

フレームワーク

- フレームワークはベストプラクティスとなる実装も多い
- ベストプラクティスとは言えない実装も多い

ベストプラクティス

CERT Top 10 Secure Coding Practices

- CERTのセキュアコーディング標準準拠より、この一般的コーディング指針が重要
- **独自のセキュアコーディング標準の構築**が必要

OWASP Secure Coding Practices

- チェックリスト形式で現在のセキュリティ対策の状態をチェック可能
- 必ずしも全てのベストプラクティスを全てのプロジェクトに適用する必要はない

OpenSAMM

- コードのみでなく「仕組み」としてセキュアな開発を実行する組織を作る方法論
- 能力成熟度モデルであるため導入し易い
- **セキュリティは文化**であり、段階的にセキュリティ文化を構築可能

お問い合わせ

セキュア開発/プログラミング・ソースコード検査のお問い合わせは

エレクトロニック・サービス・イニシアチブ

http://www.es-i.jp/

info@es-i.jp