リスクとセキュリティ

エレクトロニック・サービス・イニシアティブ

現在、リスクの分析が無い・不十分であることが、
セキュリティ問題の根本的な原因となっている

〇〇攻撃対策、といった対策には
全てのリスク対策が列挙されていない事が多い

簡易なリスク分析でもリスク分析を実施すれば
十分に対応することが可能

リスク

識別していないモノに対応することはかなり困難

「リスク」とは不確実性

「リスク分析」とは不確実性の識別

「リスク対策」とは確実にする事

全体のリスク

全体の不確実性の大きさ ＝

（不確実性1 ＊ 発生頻度1）＋

（不確実性2 ＊ 発生頻度2）＋

…

不確実性の結果が大きいと大きなリスク

不確実性の発生頻度が多いと大きなリスク

情報セキュリティ対策

「セキュリティ対策」とは不確実性の管理

目的：リスクを許容範囲に抑えて
情報システムを利用できるようにする

セキュリティを管理するには

管理するにはまず
不確実性（リスク）を“全て”識別

不確実性の
“結果”と“発生頻度”
の両方を識別

セキュリティを管理するには

リスク分析で不確実性を識別する

識別していない不確実性には対応できない

リスク分析はセキュリティ管理に必須

リスク分析の目的

不確実性に対して

ホワイトリスト型対策

○ システムが**正しく動作する事**を確実にする

× ○○**攻撃ができない**様にする

ブラックリスト型対策

リスク管理（＝セキュリティ管理）

識別

排除・緩和・移転・許容

管理

リスク（不確実性）は完全に
排除できるモノではない

セキュリティ問題
はこれらのどれか
の問題が原因

リスク管理（＝セキュリティ管理）

〇〇問題が発生

↓

〇〇対策を実施

これだけでは
「セキュリティ管理」
とは言えない

リスク分析・対応がないと
リスク（不確実性）対策が
不十分となる場合が多い

例：プレイスホルダによるSQLインジェクション対策

識別子はプレイスホルダでは保護できない

残存リスク

パラメーターはプレイスホルダで保護されている

不十分

sql_exec(“SELECT $mycol FROM table WHERE name=$1”, [$name])

識別子をエスケープしても“内容”の保護ができない

残存リスク

プレイスホルダでは“内容”の保護ができない

残存リスク

例：プレイスホルダによるSQLインジェクション対策

残存リスク

不十分

〇〇問題が発生

↓

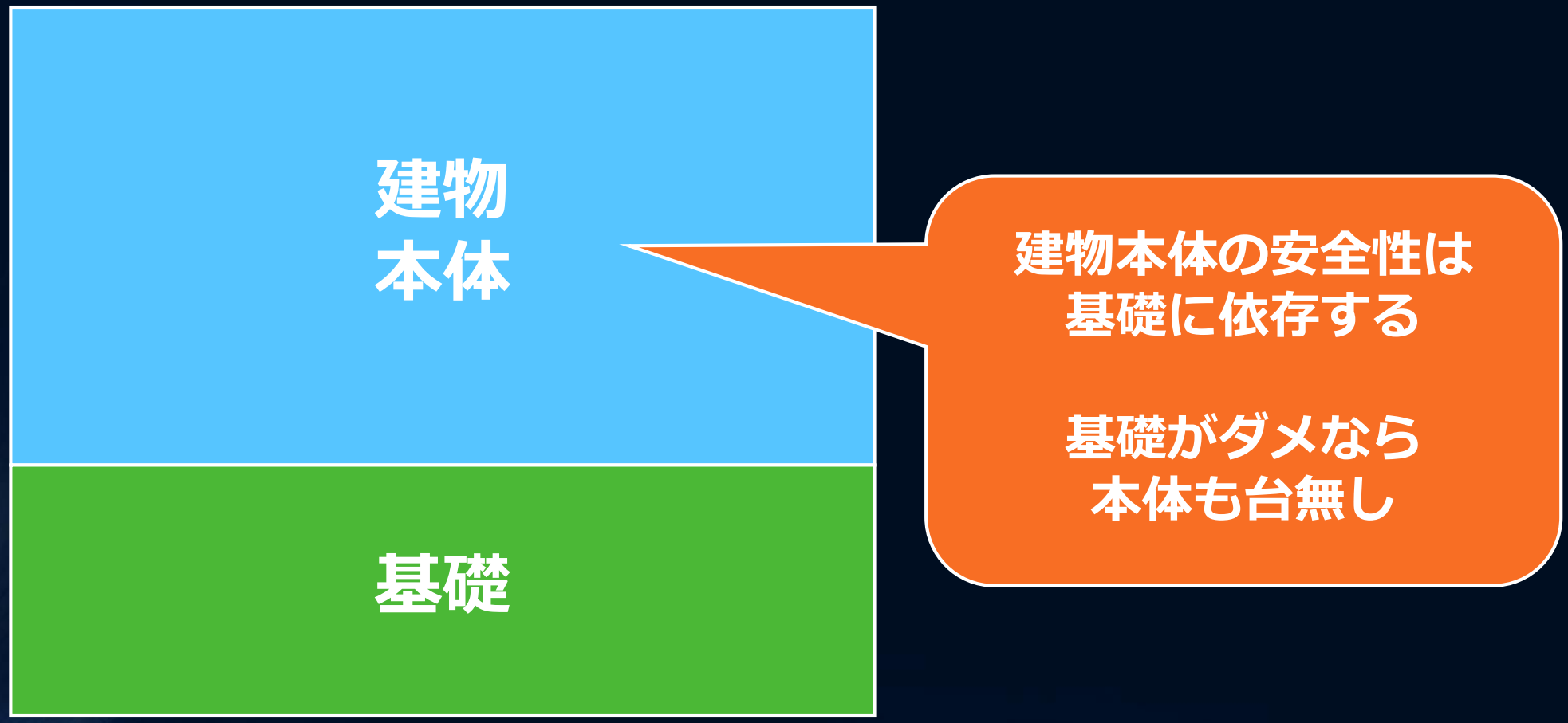
〇〇対策を実施

危険な残存リスク
だらけに・・・

残存リスク

構造とセキュリティ

建造物の構造



本体が立派でも基礎がダメなら台無し

https://openphoto.net/gallery/image/view/12091

ソフトウェアの構造

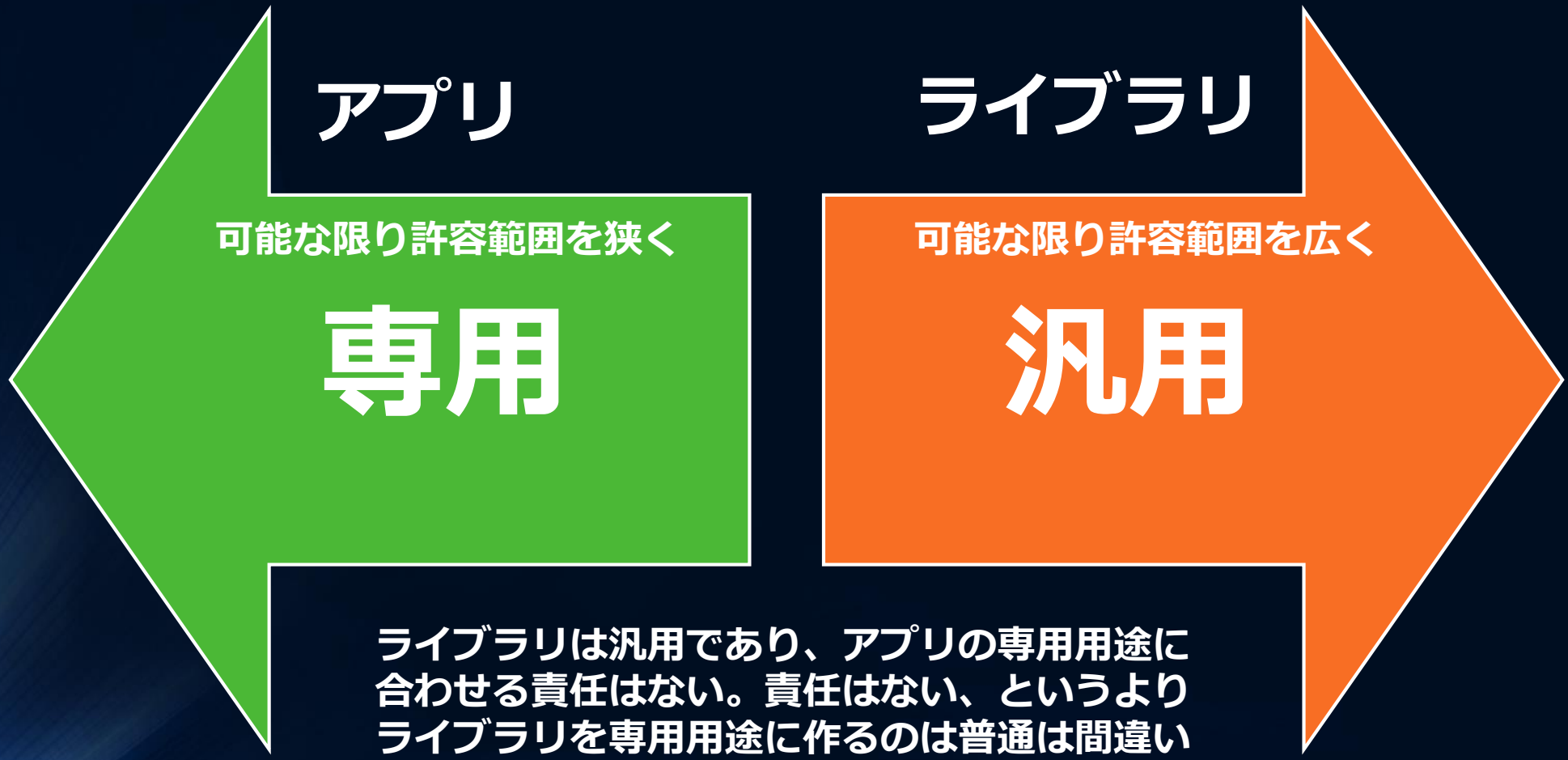
アプリ本体の安全性は
入出力に依存する

入出力がダメなら
本体も台無し

本体が立派でも基礎がダメなら台無し

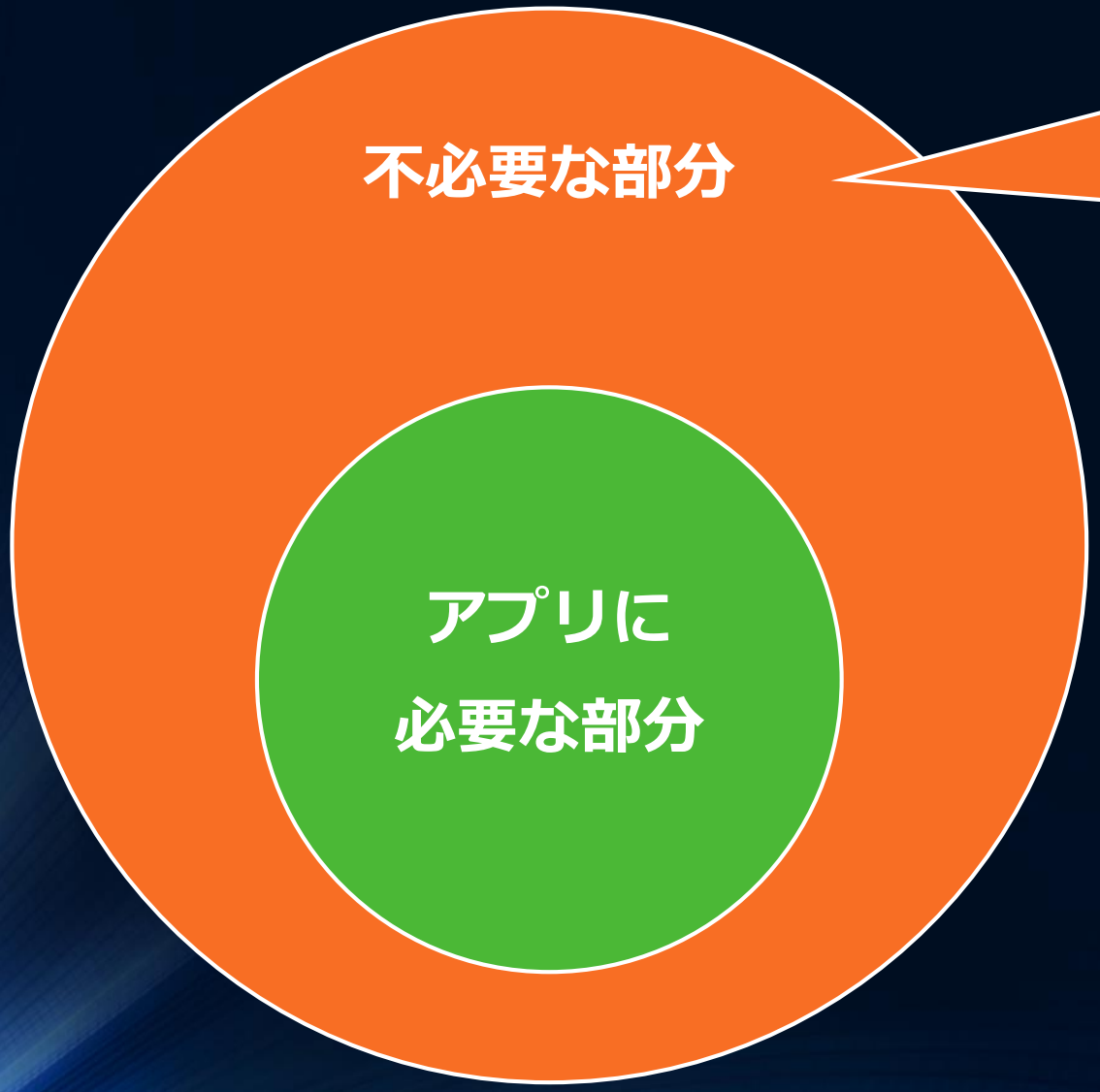
アプリとライブラリ

アプリとライブラリは性質が異なる



ライブラリは汎用であり、アプリの専用用途に
合わせる責任はない。責任はない、というより
ライブラリを専用用途に作るのは普通は間違い

ライブラリ＝可能な限り許容範囲を広く＝リスクだらけ



不確実性を
増すだけ

“汎用”のライブラリ任せのセキュリティは不確実性（リスク）を増す結果となる

ライブラリは“使えば良い”というモノではない

確実性の確保は“専用”であるアプリの責任

役立つ基礎知識

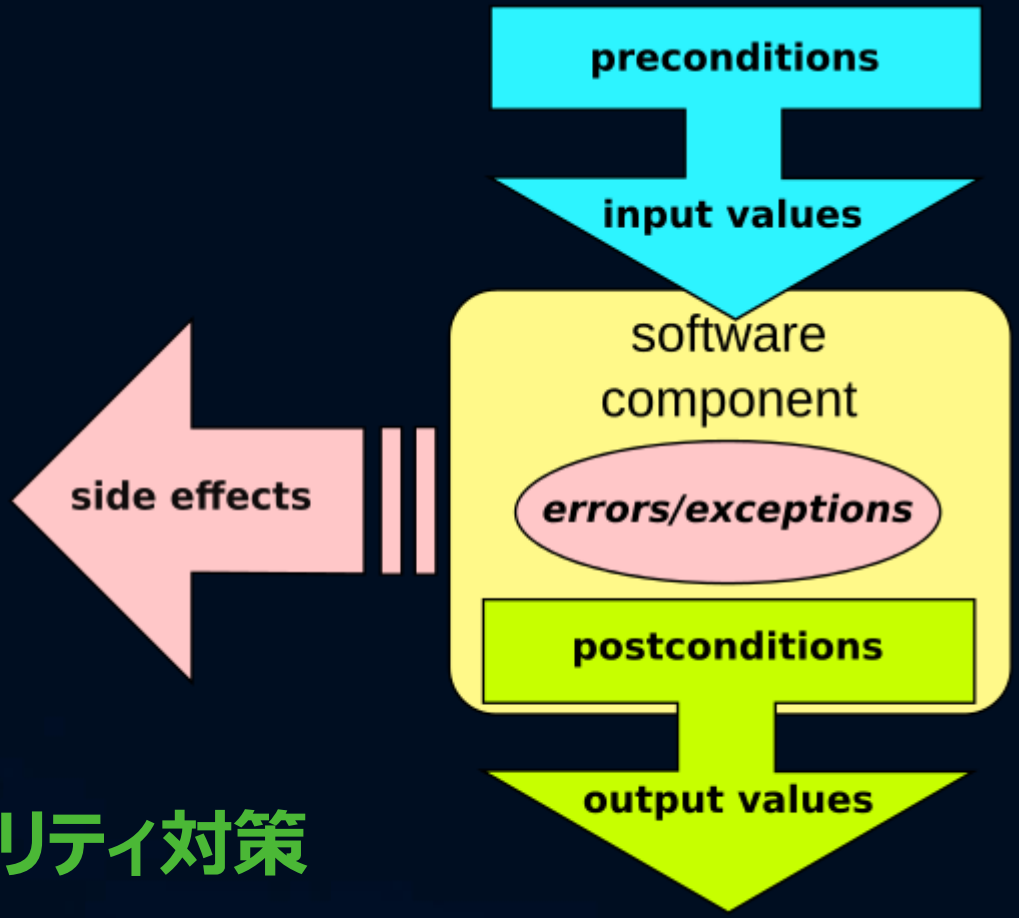
形式的検証

- **プログラムが正しく実行可能か検証する手法**
  - 論理的に検証 – 数学的に検証。正確だが困難
  - 実証論的に検証 – 総当たりで検証。検証パターンが問題

- **「組み合わせ爆発」（＝不確実性爆発）**
  - 総当たりで検証する場合にプログラムが取る状態が増えすぎる状態
  - 処理不可能な大量の組み合わせが簡単に発生する
- **状態が増え過ぎないようにする工夫が必要**
  - **妥当でない入力データは状態が増え過ぎる原因**

契約による設計/契約プログラミング

- 全ての入力をバリデーション
- 全ての状態をバリデーション
- 全ての出力をバリデーション



- 運用時にバリデーションを省略
  - 運用時に残すバリデーションがセキュリティ対策

正しく動作させるプログラムの基本原則

ゼロトラスト

フェイルファースト

ホワイトリスト

セキュアコーディング

境界防御

多層防御

全て重要な基本原則

セキュリティの６要素

機密性

完全性

可用性

信頼性

真正性

否認防止

これらに対する不確実性を排除・緩和・移転・許容する

論理検証

正しく動作する
⇒ 妥当なデータ

p ⇒ q 、q ⇒ p

必要十分条件が成り立つか検証

前提条件が重要

妥当なデータ
⇒ 正しく動作する
但し、ロジック/出力が
正しい

バリデーション

バリデーション済み
≠
安全

検証による「信頼」は「検証の範囲内」

「検証の範囲“外”」は残存リスク

未検証のモノは信頼できない

検証の省略には「保証」が必要

リスク分析

簡易なリスク分析が有用

- **リスク分析は手間が多い**
  - 全てのリスクを列挙
  - 全てのリスクを分析（STRIDE、DREAD、マトリックスなど）
  - **リスクの列挙は比較的容易**

- **リスクを列挙するだけでも有用**
  - 簡易な方法でも「**全てのリスクを列挙**」する方がよい
  - **リスク識別漏れがあると、致命的な脆弱性を残す**
    - 例1：ブラックリスト対策はホワイトリスト対策と変わらないセキュリティ対策
    - 例2：SQLインジェクション対策はプリペアードクエリ・プレイスホルダだけで十分
    - 例3：文字エンコーディングの検証はアプリに必要ない
    - 例4：入力検証はセキュリティ対策ではない

リスク分析の目的と手段

目的：情報システムが正しく動作することを確実にする

手段：全ての不確実性を識別･分析する

「目的」と「手段」を取り違えない！
取り違えると正確＆効率よいリスク分析が困難に！

リスク分析を容易にする手法

- システムの構造に合わせて部品に分割する
  - ソフトウェア基本構造は「**入力処理　→　ロジック処理　→　出力処理**」
- 上流の部品からリスク分析を行う
- 部品ごとにリスクを識別＆対応、下流の部品の前提条件とする
- ロジック処理も適当な大きさに分割し、部品ごとにリスクを分析

全体をカバーする抽象的なモノを分割して分析

分割したモノは上位のモノを全てカバーする

列挙型のリスク分析例

- **利用者が間違った入力を送信する**
  - 保存／処理できてしまう入力データ
    - 機密性／完全性／可用性／信頼性／否認防止を失う
      - 保存できる場合、間接攻撃状態に陥る
        - 保存データを利用する際に攻撃と同じ状態に陥る
          - インジェクション攻撃状態になる
          - DoS状態になる
          - システム内部情報の暴露が可能になる
          - 不完全なデータが保存される
        - 保存データが利用される場合にエラーとなる
          - DoS状態になる
          - システム内部情報の暴露が可能になる
          - 不完全なデータが保存される
        - 不完全なデータが保存される
          - この項目の上位項目の「利用者が間違った入力を送信する」に戻る。

攻撃者からの攻撃でなくても「未検証入力」があると多くのリスクが発生する

リスク分析の目的

システムが正しく動作することを確実にする

これが分析木のルート

列挙型のリスク分析例

- **利用者が間違った入力**
  - 保存／処理
    - 機密性
      - 保存
        - 保存データ
          - インジェクション
          - DoS状態になる
          - システム内部情報の暴露が可能になる
          - 不完全なデータが保存される
        - 保存データが利用される場合に
          - DoS状態になる
          - システム内部情報の暴露が可能になる

未検証入力が余計な
不確実性の発生源

残存リスク
に注意！

入力検証で不確実性を排除
次の部品のリスク分析の“前提条件”とする

データバリデーションがあると

- **利用者が間違った入力を送信する**
  - 入力処理の入力データバリデーションで妥当でないデータを排除する
  - ロジック処理の論理データバリデーションで妥当でないデータをエラーとする
  - 出力処理のバリデーション/エスケープ/セキュアなAPIでフェイルセーフ対策する

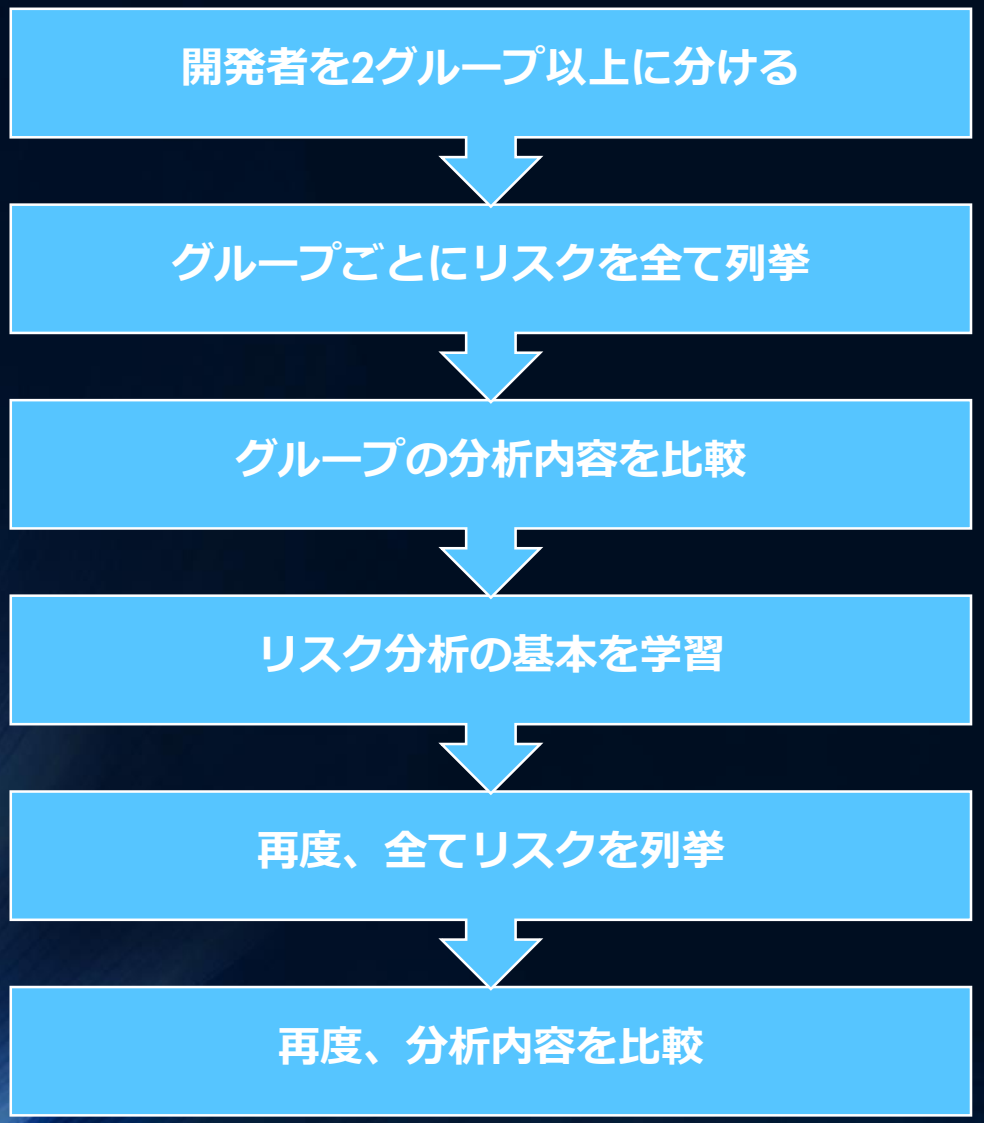
データバリデーションがある、を前提条件にするとリスク分析が飛躍的に簡素化可能

残存リスクに注意！！

リスク分析をいつ実施するのか？

- リスク分析を実施したことが無い場合、必ず実施する
  - 前提条件が無い場合のリスク分析も必ず実施
  - 徹底的に「残存リスク」の検証・確認を実施
- 同じモノに何度もリスク分析を実施する必要性は低い
  - ただし、“定期的”なリスクの再分析は必須
- 新しく利用する仕組みにはリスク分析が必須
  - 言語/フレームワーク/ライブラリが変わる場合

オススメの勉強法



新人/新入研修では必ず実施

ISO 27000では定期的なリスクの再分析を要求

前提条件の再分析は見落としがちなので注意

「残存リスク」の見落としは多いので注意

お問い合わせ先

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