PHPにない セキュリティ機能

無い機能と在る機能
エレクトロニック・サービス・イニシアチブ

自己紹介

- 氏名：大垣 靖男（おおがき やすお）
- 職業：
  - ~~PHPコミッター~~
  - エレクトロニック・サービスイニチアチブ代表取締役社長
  - 岡山大学大学院非常勤講師/PHP技術者認定機構顧問など
  - Webサイト構築関連のコンサルティングなど
    - テクニカル・セキュア開発（開発体制・コード検査・ツール販売・セキュリティパッチサービス）など
- ネット
  - http://blog.ohgaki.net/
  - yohgaki – Facebook/Twitterなど
  - yohgaki@ohgaki.net

未修正、未公開の危険な脆弱性解説を期待していた方、それは有り得ません。ご了承ください。

内容

- 既知の脆弱性
- 過去の脆弱性
- セキュアプログラミング

注意事項

このプレゼンは
ライトニングトークが
30分続きます！

実際にないセキュリティ機能

何が無いのか？
知れば自分で対策できる

後半のセキュアプログラミングを実践すれば、
自分で何が無いのか見つけられる

セッションモジュールの脆弱性

- HTTPセッションはWebアプリケーションセキュリティの要なのでロックソリッド？！
- 実は結構いい加減
- そもそもクッキーの仕組みがいい加減
- そしてセッションマネージャーの実装もいい加減

セッションモジュールの脆弱性その1

セッションアダプション

セッションアダプション

- ここで言うセッションアダプションとはクラッシックなアダプティブなセッションマネージャーの事
  - 未初期化のセッションIDを有効なセッションIDをして受け入れる脆弱性
- PHP 5.4のセッションモジュールには use_strict_modeが無い
- つまりアダプティブなセッション管理

ブラウザのクッキー管理もいい加減

- いい加減なクッキー管理により**消えないクッキーを設定可能**
- クッキーのdomain、path、secure、httponly属性の取り扱いがブラウザによって異なる
- 設定の順序によって分けの解らない動作をするブラウザもある
- 比較的最近のRFC改定によりブラウザが最も優先順位が高いとするクッキーのみが送信される
- 昔の動作であれば、同じ名前のクッキーが複数送られてきたのでサーバー側で不正なクッキーの検出が可能であったが、**現在は検出不可能**

ブラウザのクッキー管理もいい加減

- 更に悪いことに今のRFCだと、一つのレスポンスに同じクッキー名のSet-Cookieヘッダは一つのレスポンスしか返せない（ことになっている）
  - 知る限りところ複数かつ同名のSet-Cookieヘッダを無視するブラウザは知らない（今は無視するかも？）
- つまり、以下のようなオフェンシブクッキーを削除するコードはRFC的には違反

```
setcookie('foo', '', 1, "/");
setcookie('foo', '', 1, "/myapp/");
setcookie('foo', '', 1, "/myapp/login/");
setcookie('foo', '', 1, "/myapp/login/reset_cookie/");
```

セッションモジュールの脆弱性その2

古いセッションの削除

不正確なセッション削除

- セッションアダプションと同じく10年来知られている脆弱性

session_regenerate_id()

session_regenerate_id () の動作

- オプション引数なしでは古いセッションを削除しない
- session_regenerate_id(**true**)は**直ちに古いセッションを削除**する

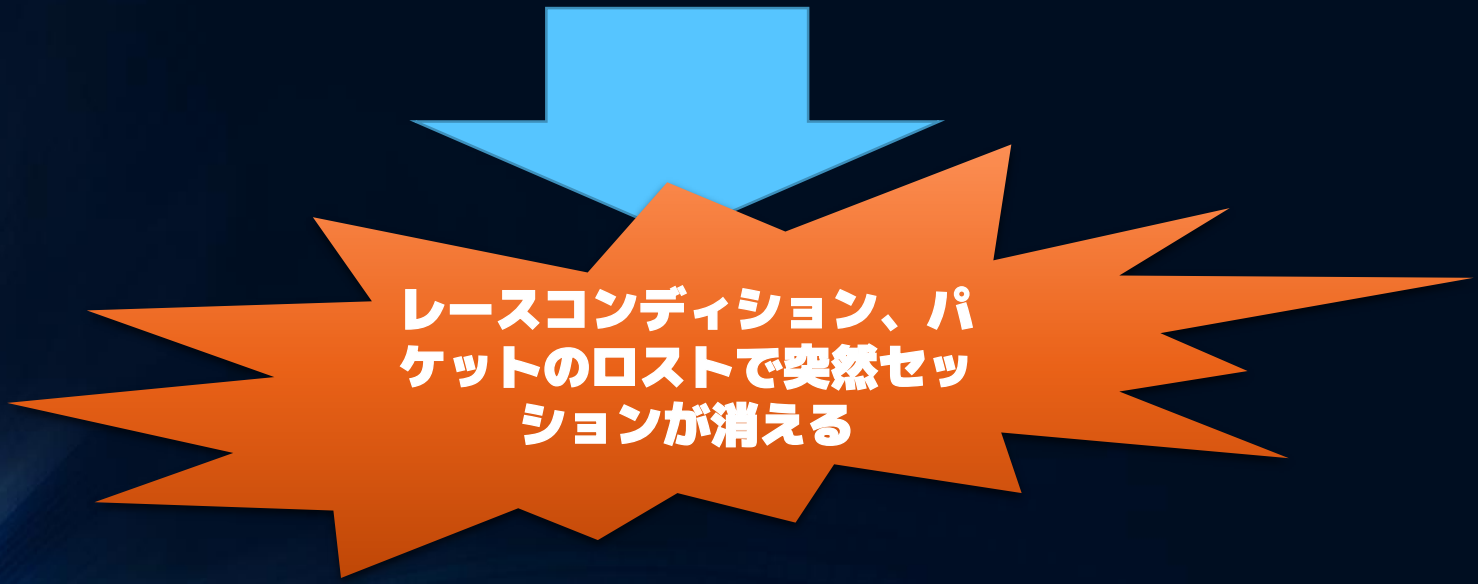
直ちに古いセッションを削除！

これで万事OK！

これで万事OKではない

直ちに古いセッションを削除はNG

- WebサーバーとWebクライアント（ブラウザ）は非同期
- Webクライアントは複数接続を利用
- ネットワーク環境は不安定（モバイル）



不正確なセッション削除

- セッションアダプションと同じく10年来知られている脆弱性

```
session_regenerate_id();
```

10年以上前にこの関数が追加された時、
セッションが消えると困るのでデフォルト
では削除しない仕様に変更

**GCに任せておけば、その
うち消える**

GC任せのセッション削除の問題

- PHPのセッションGCは「確率ベース」
- トラフィックの少ないサイトではなかなか削除されない
- 攻撃者がセッションハイジャックしている場合、新旧両方のセッションが有効

この問題への取り組み

- ユーザーランドで対応が可能
  - 削除フラグ＋タイムスタンプで確認するだけ
  - 詳しくは「間違いだらけのHTTPセッション管理とその対策」
    - http://blog.ohgaki.net/design-bugs-in-http-session-management
- 実はPHP5.6用のパッチは準備済みだった
  - ここ数年間、セッションIDの削除がGC任せである問題に取り組んでいる
  - 提案したが「**この問題とセキュリティ対策の本質**」を理解されず却下
  - 現在は理解が進みPHP7には入るかも知れない

不正なスクリプト/ファイルの実行防止

include ($_GET)

- PHPは埋め込み型言語なので

Include('/etc/passwd');　// OK!!

パスワードファイルの
中身が出力される

Include('/www/html/upload/evil.gif');　// OK!!

イメージファイルの中の
PHPスクリプトの実行もOK

include ($_GET)

- 他の言語で同じようなことをする“非”埋め込み型なので

Include(‘/etc/passwd’);　// NG!!

確実にパースエラー

Include(‘/www/html/upload/evil.gif’);　// NG!!

単純な画像ヘッダーのチェックなら回避できるが、まともにバリデーションしていればパースエラー

PHPのinclude文は読み込み・実行に脆弱

- 過去に数年間にわたり様々な改善提案
- “非”埋め込みモード：他の言語並みに安全
- 拡張子ホワイトリスト：.phar, .phpのみ実行許可
- **スクリプトホワイトリスト**：実行可能なスクリプトを指定

今はココです。開発者うけも比較的よくこの提案なら承認されるかも。

Opcacheにプリロードするので実行速度も向上する

PHP7のタイプヒント

PHP7のタイプヒントには任意精度数値がない

- データ型は単純に指定すれば良い、物ではない

OWASP Secure Coding Practices(v2) – Input Validation

Validate for expected data types

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

整数型のデータ型には、signed/unsignedがありchar/byte/short/word/int/longなどがある
浮動小数点型にも通常の倍精度のみでなく、四倍精度もある

32ビットCPUのPHPは符号付32ビット整数、64ビットCPUは符号付64ビット整数

PHP7は任意精度整数、浮動小数点型のタイプヒントがない

- 特に32ビットCPUのPHPは符号付32ビット整数しかサポートしないので影響が大きい

DBのIDは普通は64ビット整数

```
function getMyTableByID(int $id) {
  // データベースからレコードを取得
}
```

JSONデータ？numericに精度指定は無い

タイプヒントエラーで例外が発生

- 実際にどのような例外になるのか、現在議論中
- おそらくthrowableを継承したタイプエラー例外

例外をキャッチしないと
スクリプトの実行が終了

重要：データ型は指定すれば良い、という物ではない

- PHPがサポートするデータ型（整数、浮動小数点）の範囲を超えるデータにはint/floatタイプヒントを使ってはならない
- 年齢や月など確実に範囲内に収まるデータにのみint/floatタイプヒントを使う
- 全ての例外・エラーはキャッチする
  - PHP7のZPPではデータ型の範囲内かチェックしエラー（例外でなく、エラーです。今のところ）が発生する

整数オーバーフローを検出できない

- Webシステムの入力は基本「文字列」
  - ブラウザ
  - データベース（LDAPなどを含む）
  - JSON/XML
- 大きすぎる整数はキャスト後にPHP_INT_MAX/PHP_INT_MINのどちらかになる
- PHP_INT_MAX/PHP_INT_MINをオーバーフローとして取り扱うと簡単

JavaScriptエスケープ

PHPはWeb用言語なのに JavaScriptエスケープ関数が無い

- 無いものは仕方ありません。
- 参考
  - http://blog.ohgaki.net/javascript-string-escape

LDAPエスケープ

LDAPエスケープが無い、無かった

- PHP 5.5までLDAPエスケープ関数が無い
- LDAPエスケープにはFilterコンテクストとDNコンテクストのエスケープが必要
- ldap_escape関数はPHP 5.6から
- 参考
  - http://php.net/manual/ja/function.ldap-escape.php

XPathエスケープ

XPathエスケープが無い

- XPath 1.0の定義には、そもそもエスケープ方法が定義されていない
- XPath 1.0の仕様は壊れている
- 参考
  - http://blog.ohgaki.net/xpath-query-1
  - http://blog.ohgaki.net/xpath-query-2
  - http://blog.ohgaki.net/reason-why-xpath1-0-is-broken
- XPath 2.0ではエスケープ仕様が追加された

SQL識別子エスケープ

SQL識別子エスケープが無い

- PostgreSQLモジュールにはSQL識別子エスケープ関数（pg_escape_identifier）がある
- 他のDBモジュールには無い
- DBMS自体は識別子エスケープを定義
- しかし、PostgreSQLを除いてクライアントアクセスライブラリに識別子エスケープAPIがない
- 仕様として問題

JSONエスケープ

JSONエスケープはある

- しかし、デフォルトで安全ではない
- 参考
  - http://blog.ohgaki.net/json-escape
  - http://blog.ohgaki.net/php7-json-decode-json-encode-problem

JSON Numeric

JSON Numeric型は int/floatに自動変換されてしまう

- intはオプションで文字列のまま取得可能
- floatは無理
  - PHP7用にオプションで文字列のまま取得可能にするよう検討中
- 基本的にこのような自動変換/型の強制はセキュリティ的に問題

HTMLコンテクスト エスケープ

HTMLエスケープはあるがその他コンテクスト用エスケープが無い

- OWASP ESAPIは各コンテスト用のエスケープ（エンコード）メソッドを提供
  - https://github.com/OWASP/PHP-ESAPI
  - 最近は更新されていない

htmlspecialcharsは　デフォルトで安全でない

- PHP　5.6未満は明示的文字エンコーディング設定が必要
- PHP　5.6以降はdefault_charsetが利用される
- ‘　（シングルクオート）をデフォルトでエスケープしない
- HTML5に対応していない
  - / のエスケープに対応していない。HTML5ではクオート文字なしの属性が許される

OSコマンド エスケープ

OSコマンドの完全なエスケープは非常に困難

- 多数のOS、シェル＋コマンド引数パーサー
- escape_shell_argsも完璧ではない
- 参考
  - http://blog.ohgaki.net/os-command-escape-shell-spec-command-implementation

- pcntl_execはコマンドと引数を分離して実行
  - 内部的にexecv, execveを利用
  - shell_execなどで使えるように提案中

XMLエスケープ

XML用のエスケープがない

- 無いです
- htmlspecialchars/htmlentitiesが流用可能

LIKEエスケープ

SQL文のLIKEエスケープがない

- おかしなクエリ結果が返るだけで、セキュリティ上のインパクトがあることは少ない
- しかし、**余計な結果が表示されることが問題**となる場合も
- そもそもDBMSがエスケープAPIを提供していない
- “%”、“_”をエスケープ

正規表現メタ文字

正規表現
メタ文字エスケープ関数がない

- quotemeta関数はあるがこれでは不十分
  - http://php.net/manual/ja/function.quotemeta.php
- PCRE用には専用関数がある
  - http://php.net/manual/ja/function.preg-quote.php
- mbstring用には専用関数がない
  - PCRE用が流用できる
- SQLのLIKE文と同じく、余計な結果が返ることが問題となる場合がある
- エスケープしないと正規表現ライブラリ脆弱性攻撃に利用される

入力バリデーター

使いやすい
入力バリデーターが無い

- Filterモジュールは在る
  - http://php.net/manual/ja/book.filter.php
- 入力バリデーターとして使いやすい、とは言いがたい
- フレームワークのバリデーターを使いましょう

PHPに在った脆弱性
PHPに無かったセキュリティ機能

どんな問題が在ったのか？
知れば対策できる

自分のコードのセキュリティ問題でなくても、
自分のコードで対策できる

タイプコンフュージョン

タイプコンフュージョン

- タイプコンフュージョンとはPHP組み込み関数が変数型を間違えて処理してしまう脆弱性
  - 最近、頻繁に修正されている脆弱性
- 例えば、内部関数が整数型を期待しているのに、ユーザーが配列型を渡す

```
+-Core:
+ . Additional fix for bug #69152 (Type confusion vulnerability in
+    exception::getTraceAsString). (Stas)
+- SOAP:
+  . Fixed bug #69152 (Type Confusion Infoleak Vulnerability in unserialize()
+    with SoapFault). (Dmitry)
```

PHP 7のZVAL（PHP変数の構造体）の定義

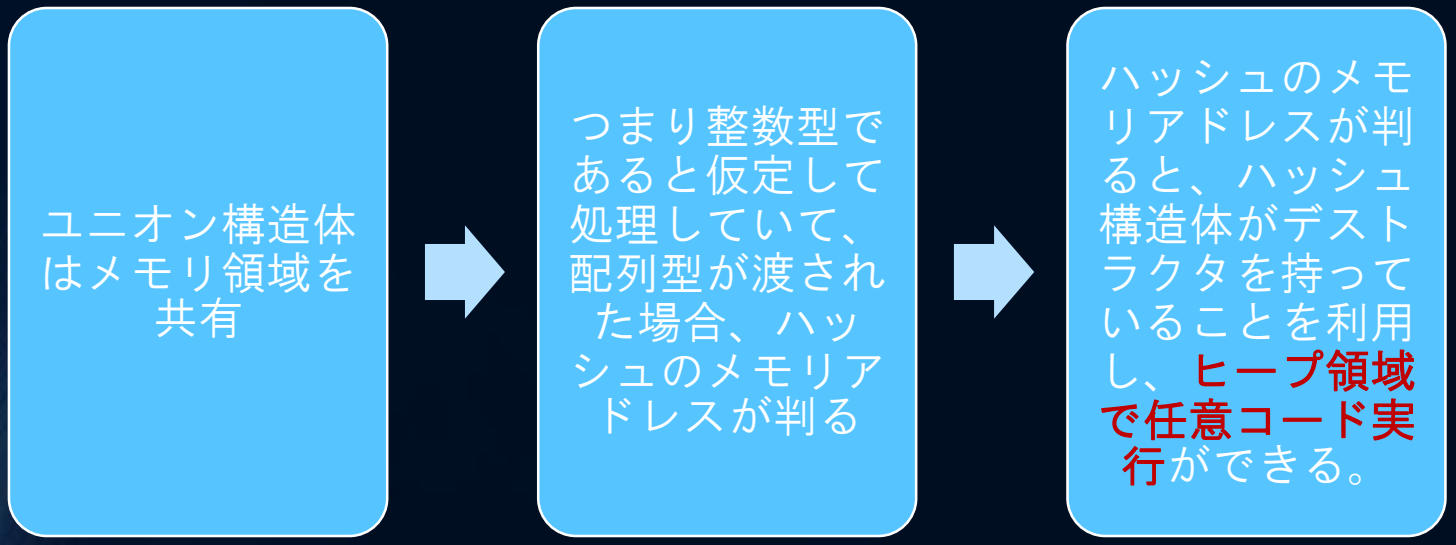
```
typedef union _zvalue_value {
        long lval;
        double dval;
        struct {
                char *val;
                int len;
        } str;
        HashTable *ht;                          /* hash table value */
        zend_object_value obj;
} zvalue_value;

struct _zval_struct {
        /* Variable information */
        zvalue_value value;                     /* value */
        zend_uint refcount__gc;
        zend_uchar type;    /* active type */
        zend_uchar is_ref__gc;
};
```

実際に変数を保存する
ユニオン構造体

これがPHP変数の構造体

タイプコンフュージョン



ただし、そのメモリに書き込める別の脆弱性が必要

タイプコンフュージョンは終わったか？

- PHPのモジュールは多様
- 多数の3rdパーティーモジュール

対応済みなので安心！

タイプコンフュージョンは終わったか？

- PHPのモジュールは多様
- 多数の3rdパーティーモジュール

対応済みなので安心？

Be Defensive!

Use After Free

Use After Free

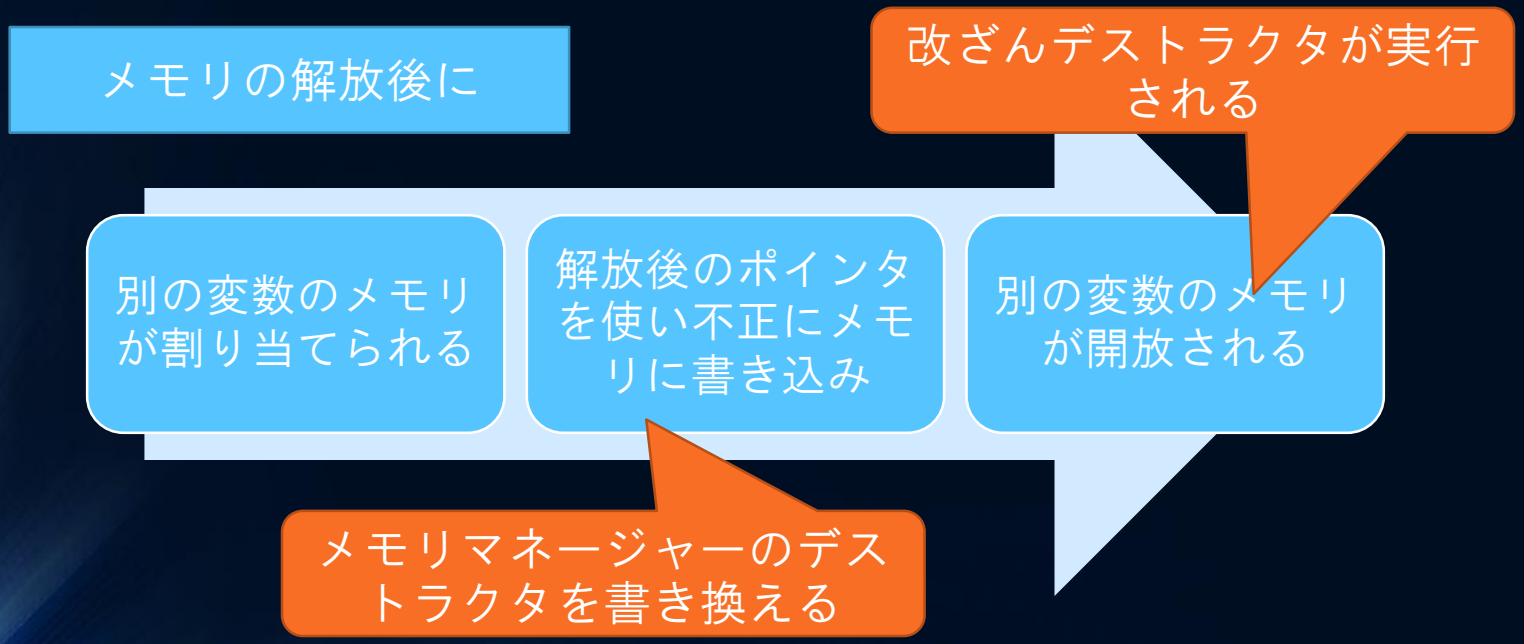
- PHP 5.4.40のNEWSより

- cURL:
  . Fixed bug #69316 (Use-after-free in php_curl related to CURLOPT_FILE/_INFILE/_WRITEHEADER). (Laruence)

- Sqlite3:
  . Fixed bug #66550 (SQLite prepared statement use-after-free). (Sean Heelan)

Use After Free

- 開放済みメモリにアクセスしてしまう脆弱性
  - 高度なメモリ破壊攻撃の一種
  - スタックオーバーフローは単純なメモリ破壊攻撃



PHPユーザーの防御策（緩和策）

出来る限り厳格な
入力バリデーションを行う

- 数値データのみなら文字列でも20バイトも無い
- 20バイト以下のシェルコードは困難
- 数字だけでは無理（ただし、制御コードなしのシェルコードは可能）

CERT Top 10 セキュアコーディングプラクティス
1. 入力をバリデーションする
全ての信頼できないデータソースからの入力をバリデーションする。適切な入力バリデーションは非常に多くのソフトウェア脆弱性を排除できる。ほぼ全ての外部データソースに用心が必要である。これらにはコマンドライン引数、ネットワークインターフェース、環境変数やユーザーが制御可能なファイルなどが含まれる。

include文に対する ヌル文字インジェクション対策

include文に対する ヌル文字インジェクション

- 例

```
<?php

include(“/path/to/any.file¥0.png”);
```

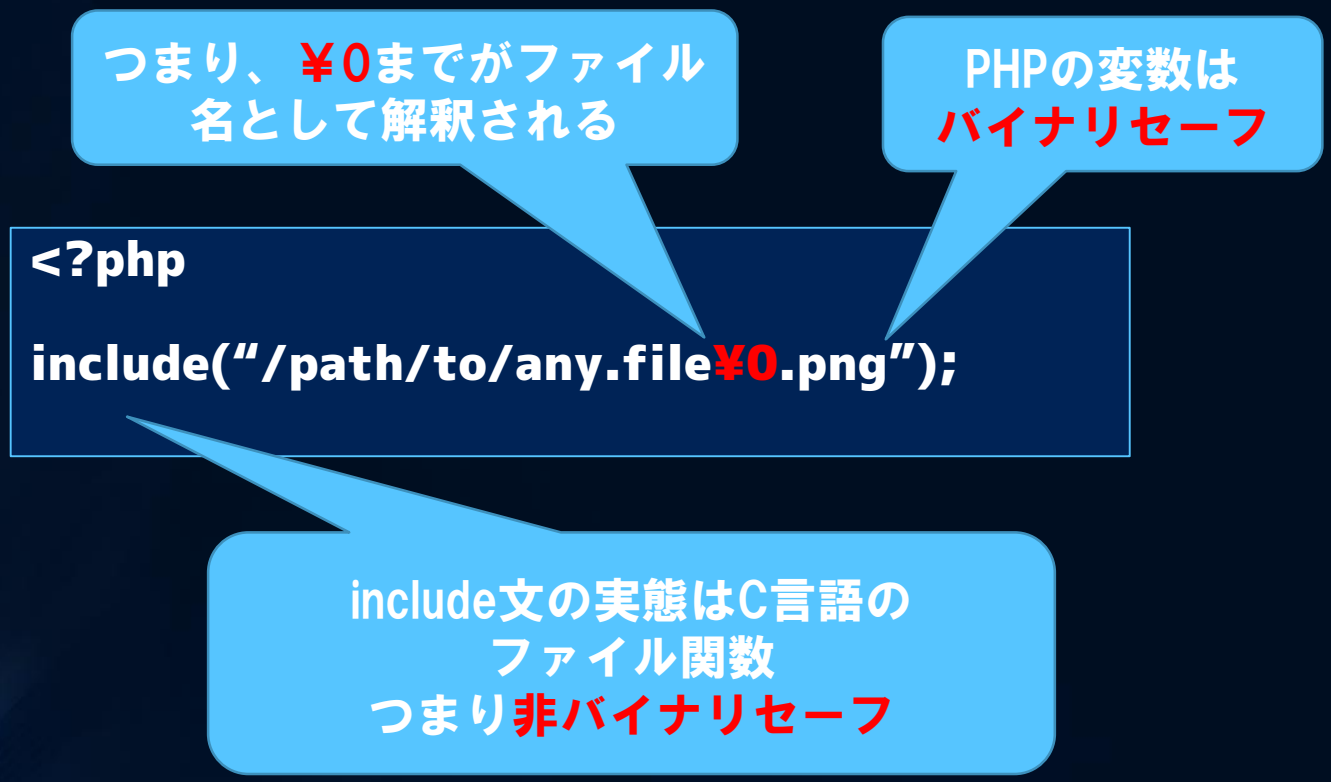
include文に対する ヌル文字インジェクション対策？？

- 例

```
<?php
//ファイルアップロード処理のどこかで
//拡張子が画像かチェック
if (!preg_match(“.png¥z”, $_GET[“module”)) {
   die(“Do not upload other than .png”;
}
```

```
<?php
$module = $_GET[‘module’];
include(PATH_MODULE . $module);
```

include文に対する ヌル文字インジェクション対策



include文に対する ヌル文字インジェクション対策？？

```
<?php
//ファイルアップロード処理
//拡張子が画像かチェック
if (!preg_match(".png¥z", $_GET["module"])) {
    die("Do not upload
}
```

preg_matchはバイナリセーフ。¥0も普通の文字として解釈される

PHP変数はバイナリセーフ。$_GETなら%00は¥0文字として解釈される

```
<?php
$module = $_GET['module'];
include(PATH_MODULE . $module);
```

include文は非バイナリセーフ
"../../../etc/password%00.png" なら

include文に対する ヌル文字インジェクション対策

PHP5.3.4からストリームラッパーでヌル文字を不許可
http://attack/script.php%00.png
なども不許可

```
<?php

include(“/path/to/any.file¥0.png”);
```

- Security enhancements:
  . Paths with NULL in them (foo¥0bar.txt) are now considered as invalid. (Rasmus)

他の言語での対応

- PHPの後で同様の対応が行われる
- 画像処理やSSL関連脆弱性ではこのパターン（PHPの後追い）が結構多い

ヌル文字インジェクション

ヌル文字インジェクションは終わったか？

- ついこの前のリリースでも脆弱性が修正される

14 May 2015 PHP 5.4.41
. Fixed bug #69418 (CVE-2006-7243 fix regressions in 5.4+). (Stas)

PHP5.3.4から**ストリームラッパー**でヌル文字を不許可
http://attack/script.php%00.png
なども不許可

```
<?php

include("/path/to/any.file¥0.png");
```

ストリームラッパー以外での場所では対応していなかった。
例）unlinkなど

ヌル文字インジェクションは終わったか？

- ストリームラッパー以外の場所での対応

```
diff --git a/ext/standard/file.c b/ext/standard/file.c
index 708c3e2..21e1e
--- a/ext/standard/fil
+++ b/ext/standard/f
@@ -813,7 +813,7 @@
        char *p;
        int fd;

-       if (zend_parse_parameters(ZEND_NUM_ARGS() TSR     S_CC, "ps", &dir,
&dir_len, &prefix, &prefix_len) == FAILURE) {
+       if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "pp", &dir,
&dir_len, &prefix, &prefix_len) == FAILURE) {
                return;
        }
```

ZPPを拡張し"p"タイプを追加
"p"タイプはヌル文字を拒否

ヌル文字インジェクションは終わったか？

- ストリームラッパー以外の場所での対応

```
diff --git a/ext/standard/file.c b/ext/standard/file.c
index 708c3e2..21e1e
--- a/ext/standard/fil
+++ b/ext/standard/f
@@ -813,7 +813,7 @@
                char *p;
                int fd;

-               if (zend_parse_parameters(ZEND_NUM_ARGS() TSR     S_CC, "ps", &dir,
&dir_le
+               if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "pp", &dir,
&dir_le
```

ZPPを拡張し"p"タイプを追加
"p"タイプはヌル文字を拒否

3rdパーティモジュールなど
ZPP拡張の"p"タイプに未対応
ならヌル文字インジェクション
に脆弱になる

ヌル文字インジェクションは終わったか？

- ストリームラッパー以外の場所、ZPPの“ｐ”タイプに未対応のケースを考慮しなければならない

OWASP Secure Coding Practices

以下の物は個別に確認しなければならない

- ヌル文字（%oo）をチェックする
- 改行文字（%od, %oa, ¥r, ¥n）
- パスを変更する文字列（../ および ..¥）をチェックする。
- UTF-8の拡張文字セットがサポートされている場合、%c0%ae%c0%ae/ など別形式の形もチェックする
  （Canonicalizeを利用し重複エンコーディングや他の不明瞭化攻撃に対応する）
  （訳注：../ や ..¥ を取り除きなさい、ではありません。単純に取り除くだけだと簡単に攻撃できます）

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

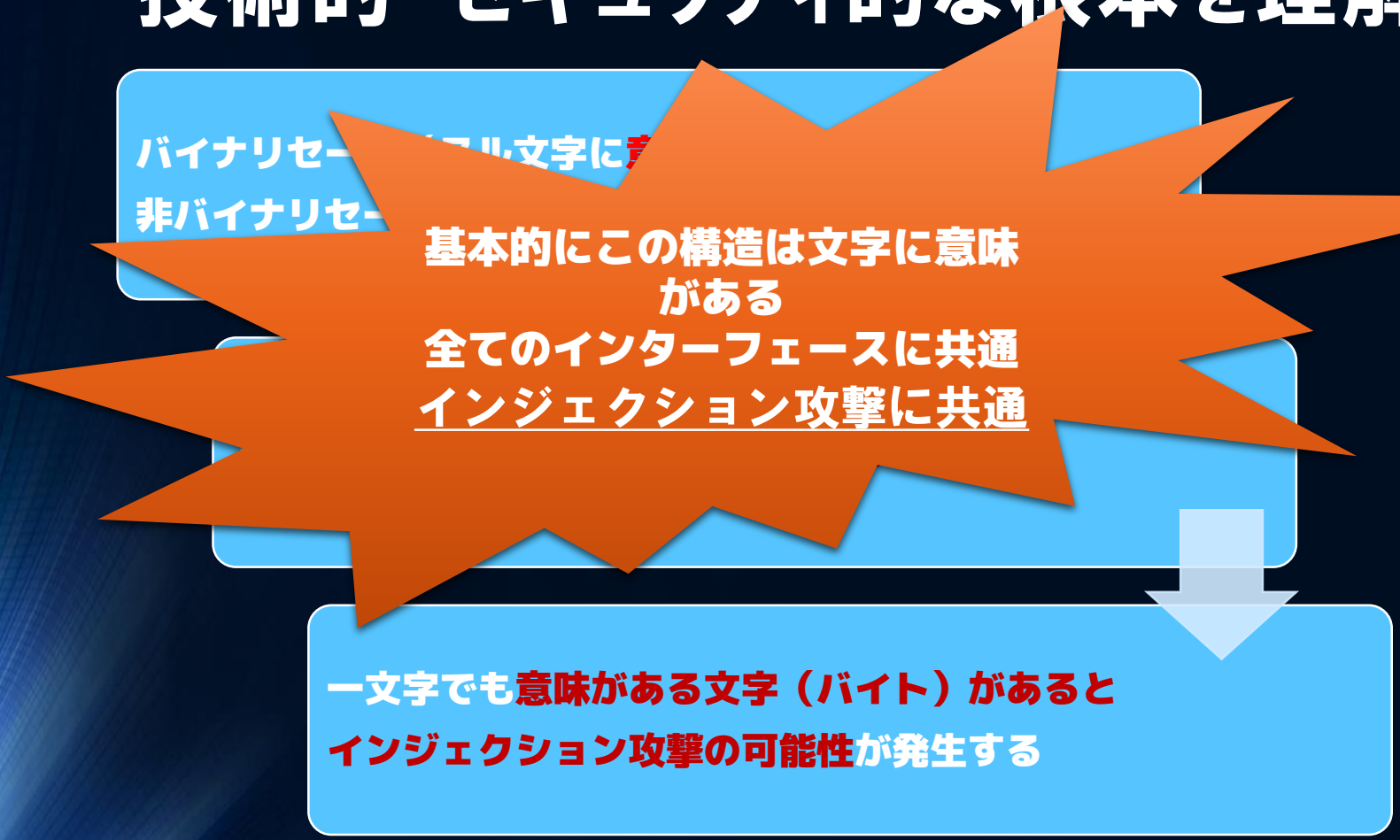
ヌル文字インジェクションの技術的・セキュリティ的な根本を理解

バイナリセーフ（ヌル文字に**意味がない**）
非バイナリセーフ（ヌル文字が**文字列終端**）

ヌル文字の解釈の違い → 処理に**相違が発生**

一文字でも**意味がある文字**（バイト）があると**インジェクション攻撃の可能性**が発生する

ヌル文字インジェクションの技術的・セキュリティ的な根本を理解



改行インジェクション

ヌル文字インジェクションに類似したセキュリティ問題

- HTTPヘッダーインジェクション
  - 別名：
    HTTPレスポンススプリッティング
    HTTPヘッダー分割
    改行インジェクション
- メールヘッダーインジェクション
  - 別名：
    改行インジェクション

HTTPヘッダーインジェクション

- header()/setcookie()関数で対策済み
- PHPからHTTPヘッダーを送る手段はheader()/setcookie()しかない（基本的には）
  - 自分でソケット作ったり、cURLでリクエストを送ったりすると、当然自分でHTTPヘッダーを取り扱うことになる
- 新しいRFCでは複数行にまたがる単一HTTPヘッダーは許可されていない
  - IEは既にこのタイプのヘッダーは無視している
- **HTTPヘッダーに改行を許さない、でOK**

メールヘッダーインジェクション

- mail関数にはメールヘッダーインジェクション脆弱性が残されている
  - https://github.com/php/php-src/pull/1273
  - https://bugs.php.net/bug.php?id=68776

[2015-01-09 09:59 UTC] yohgaki@php.net
Description:
---
mb_send_mail() parses additional headers and stores into hash. During the parse process, invalid headers are discarded.

However, mail() simply check ¥o and strip trailing ¥r¥n. Therefore, mail() is vulnerable to mail header injections via additional header parameter.

メールヘッダーインジェクション

- https://bugs.php.net/bug.php?id=68776 はセキュリティバグとして処理されていない
- PHPコミッターはどう考えているのか？
  - 入力バリデーションは当たり前
  - PHP本体だけでなく3rdパーティモジュールもある
  - そもそもPHPプログラム/ライブラリ/フレームワークの脆弱性も考慮すべき
    - 例）ZendFramework 1.xのメールヘッダーインジェクション

改行インジェクションとは

- 一行が意味を持つプロトコルに有効な攻撃手法
  - HTTPプロトコル
  - SMTPプロトコル
  - その他、**一行が意味を持つプロトコルほぼ全て**

例：memcachedインジェクション

問題の本質を知ることが、
問題への対応・解決への近道

HTTPヘッダレスポンススプリッティング、メールヘッダーインジェクションの理解は
本質の理解に比べ重要ではない

改行インジェクションの技術的・セキュリティ的な根本を理解

・一行ごとに**意味がある**

余計な改行は余計な意味を追加する
→ **余計な処理が発生**

一文字でも意味がある文字（バイト）があるとインジェクション攻撃の可能性が発生する

ヌル文字・改行インジェクションの技術的・セキュリティ的な根本を理解

基本的にこの構造は全ての
可変長インターフェースの
インジェクション攻撃に共通

固定長インターフェースの
インジェクション攻撃には
オーバーフロー攻撃が有効

C言語のバッファーオーバーフロー攻撃が有名だが、
固定長テキストデータでも同類の攻撃が可能な場合もある

セキュアプログラミング
防御的プログラミング

セキュア/防御的プログラミングのエッセンス

あらゆる脆弱性に対策できる根本的脆弱性対策を理解する

セキュアプログラミング/セキュアコーディング/防御的プログラミングの中でも重要な契約プログラミングを紹介

セキュリティ脆弱性に対する攻撃とは

システムやプログラムに対して

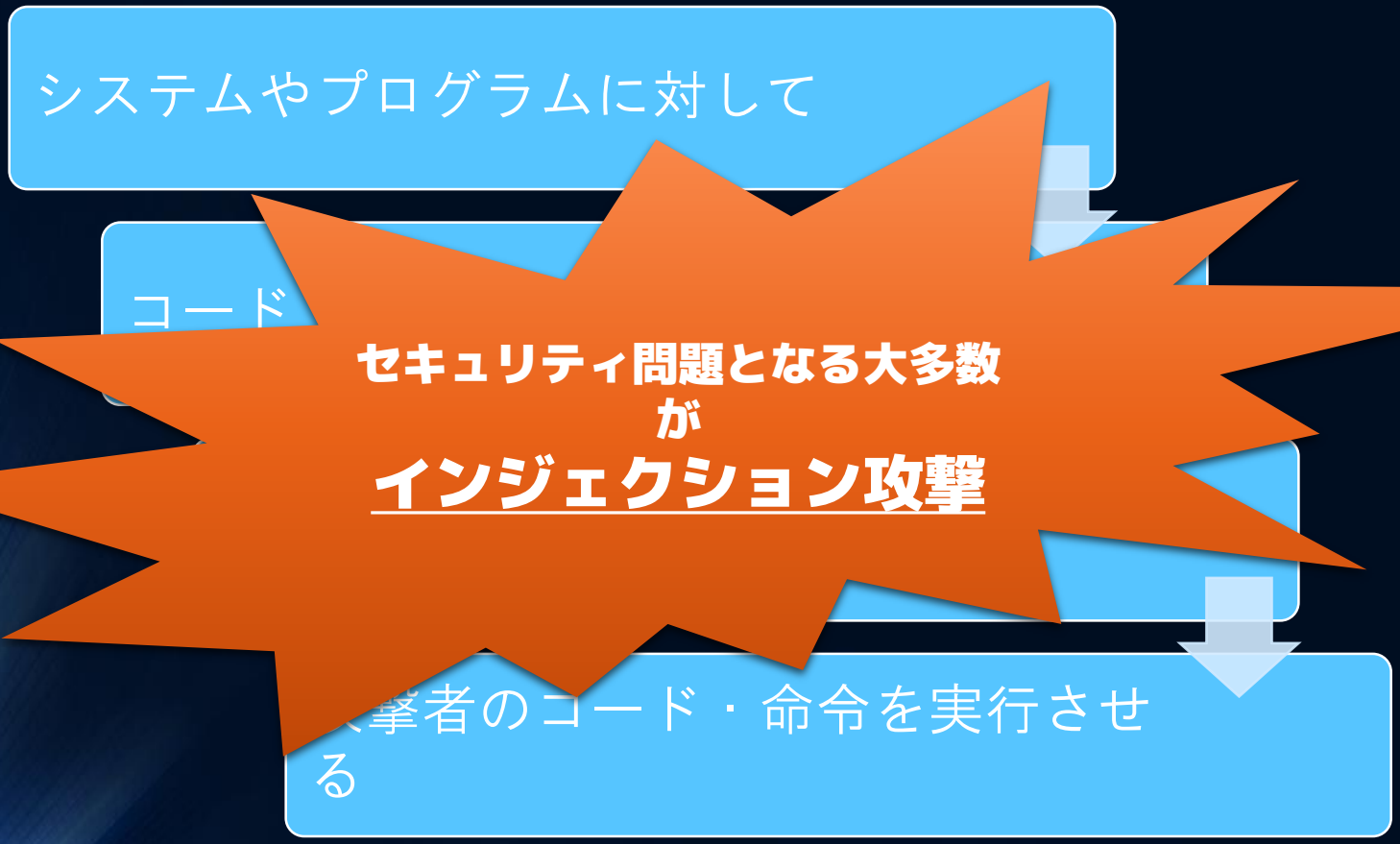
コード・命令を挿入し

意図しない動作をさせる

攻撃者のコード・命令を実行させる

※インジェクション攻撃の場合。認証/認可/DoSなどは別

セキュリティ脆弱性に対する攻撃とは



※インジェクション攻撃の場合。認証/認可/DoSなどは別

セキュリティ脆弱性に対する攻撃とは

システムやプログラムに対して

ここで防御するにはバグフリーが必要

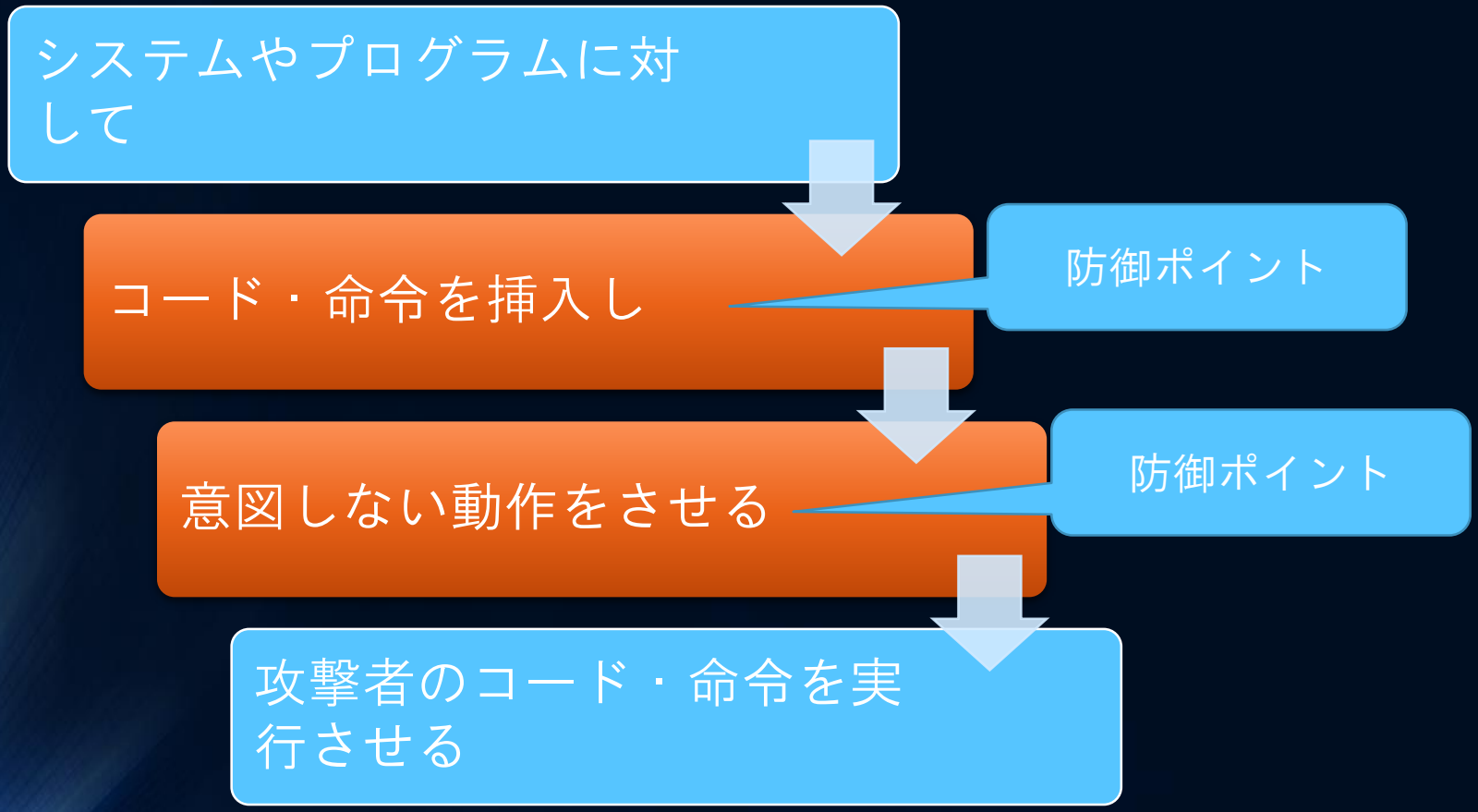
コード・命令を挿入し

意図しない動作をさせる

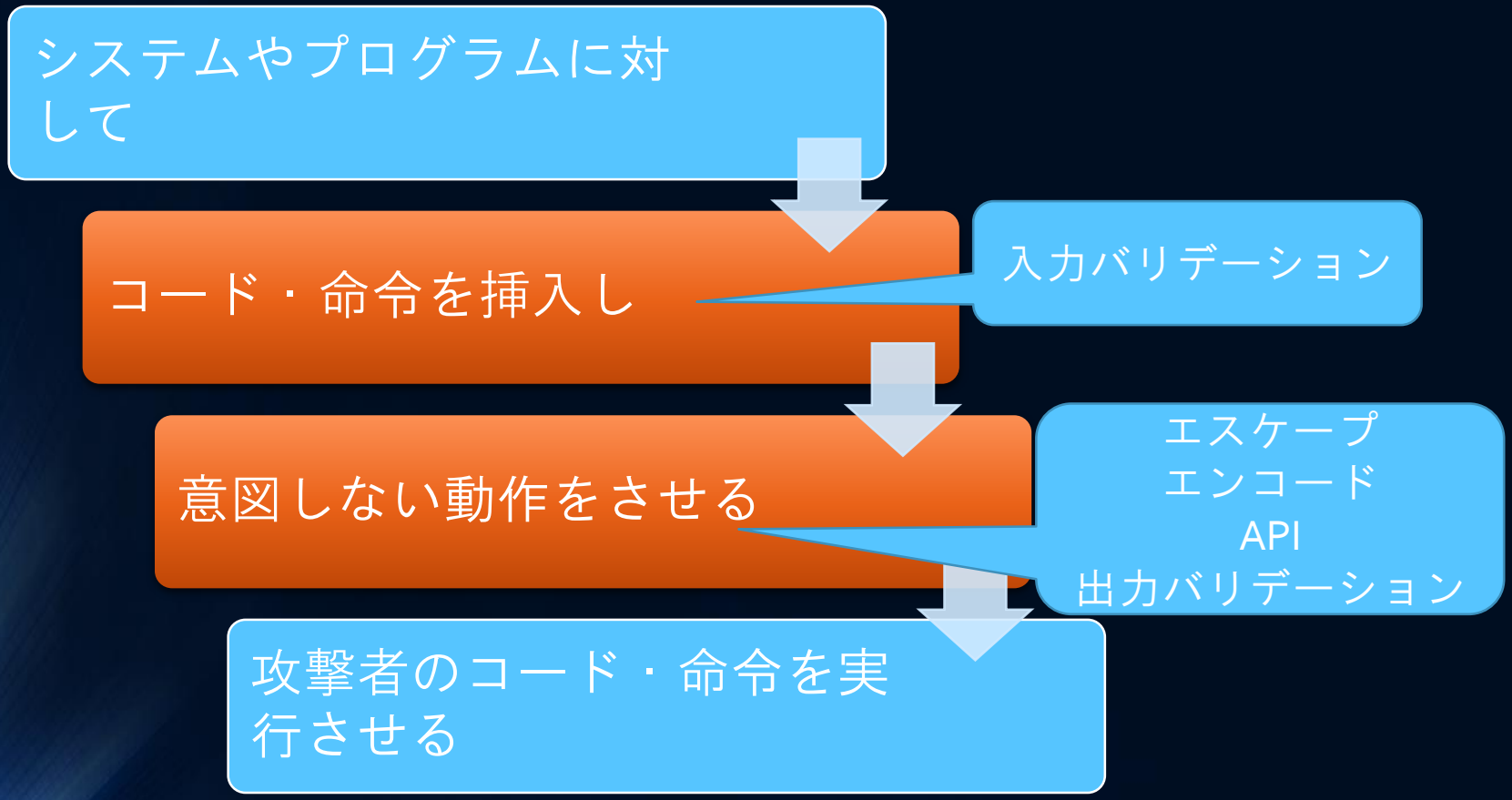
攻撃者のコード・命令を実行させる

アプリケーションの制御範囲外

セキュリティ脆弱性に対する攻撃とは



セキュリティ脆弱性に対する攻撃とは



全体最適化と部分最適化

全体最適化

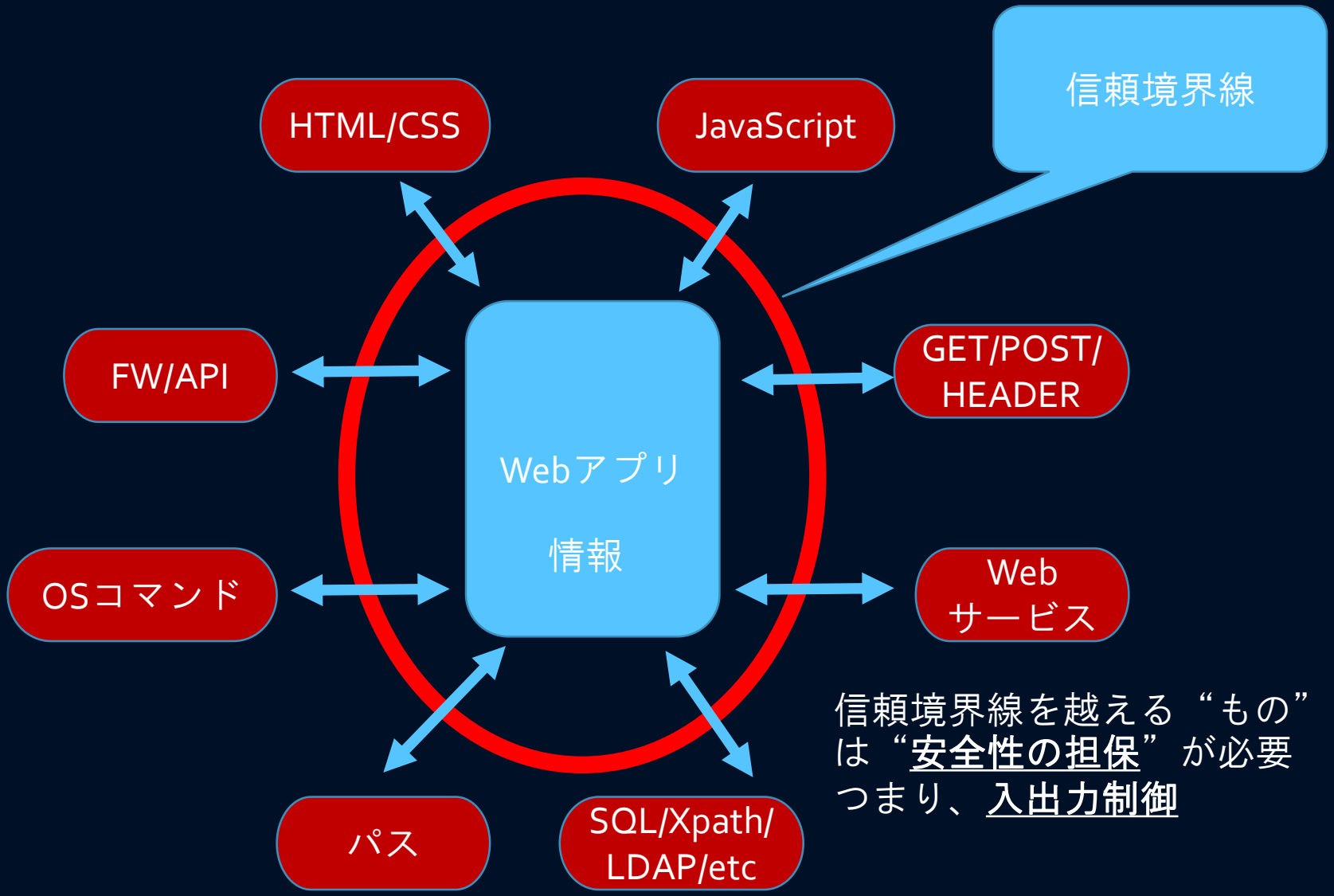
- 全体として最適化しない場合、目的が達成できない

部分最適化

- 部分最適化も重要だが、
  部分最適化では全体最適は実現しない
- 合成の誤謬
- http://blog.ohgaki.net/fallacy-of-composition-and-it-security

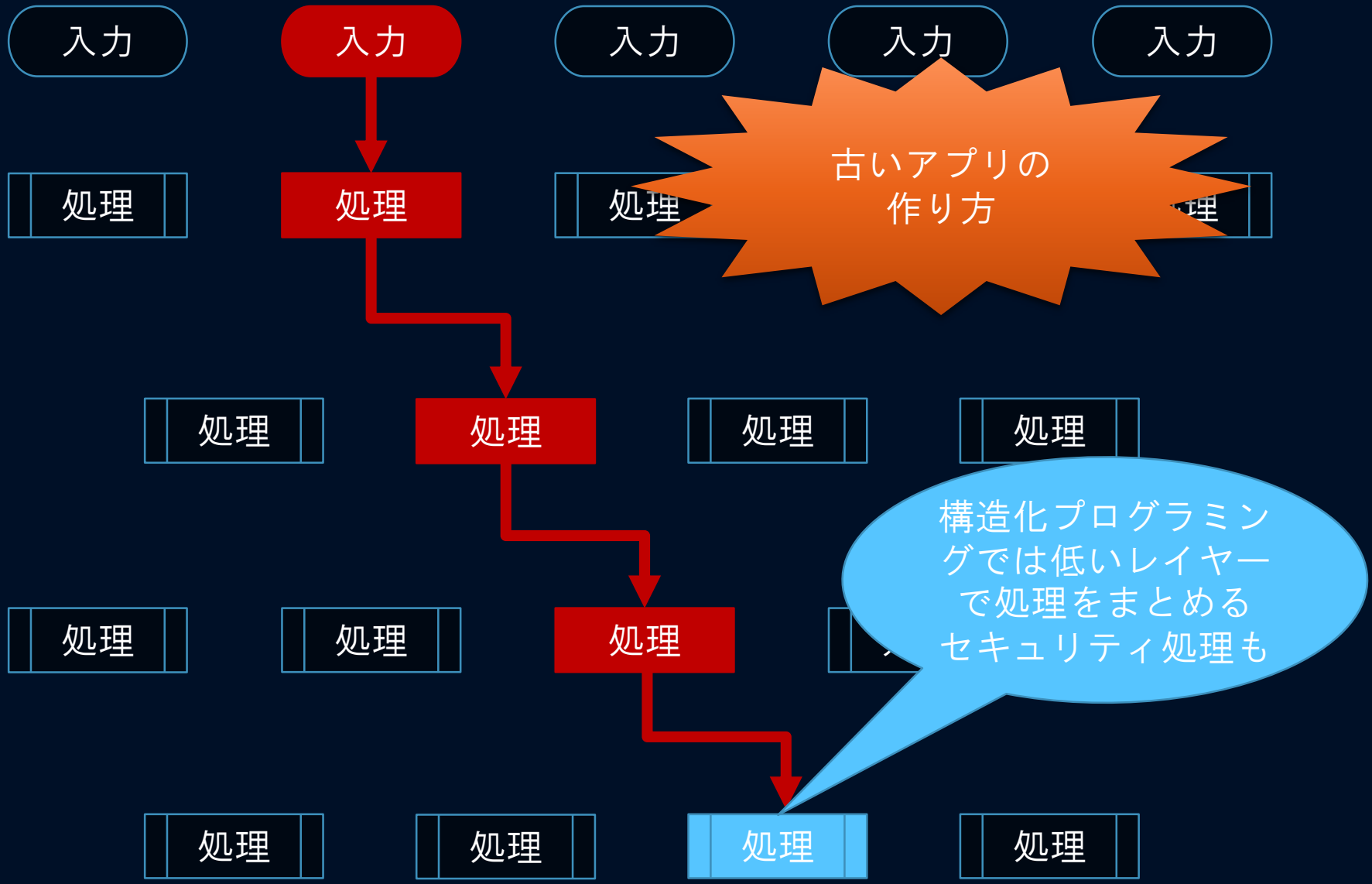
部分最適化で
全体最適化ができない理由

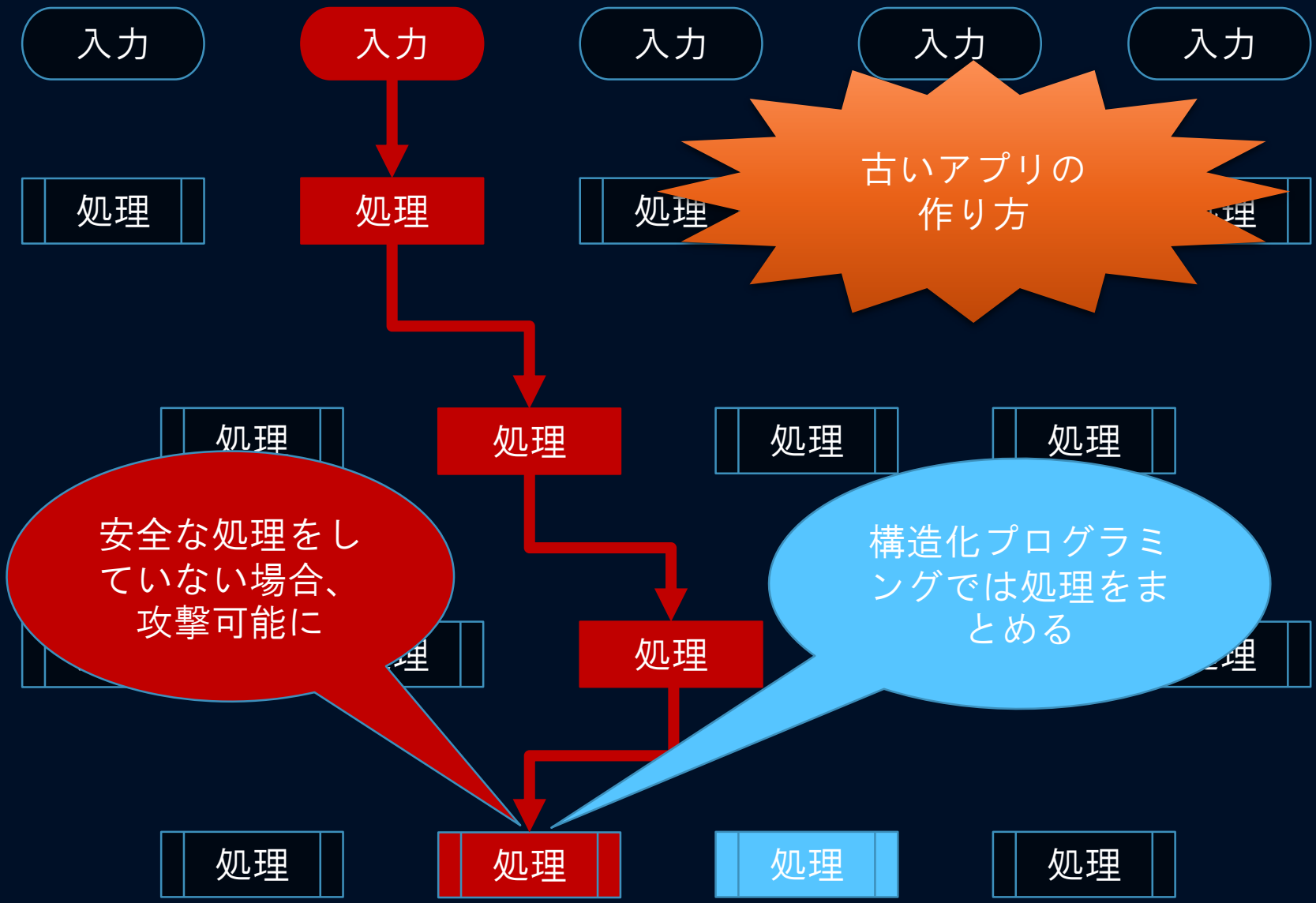
- 部分最適化の場合、アプリケーションロジックの中にセキュリティ対策が埋没する
- アプリケーションロジックのバグは発見しづらい
- しかも、変更が多い
  - アプリ自体の仕様変更、リファクタリング（継続的開発）
  - テストの主目的が「正常系」テスト
  - ライブラリ/フレームワークの更新・バージョンアップ
  - OS/低層ライブラリの更新・バージョンアップ
- **個々の対策を積み重ねても、全体的に対策した状態にならない**



信頼境界線を越える“もの”は“**安全性の担保**”が必要
つまり、**入出力制御**

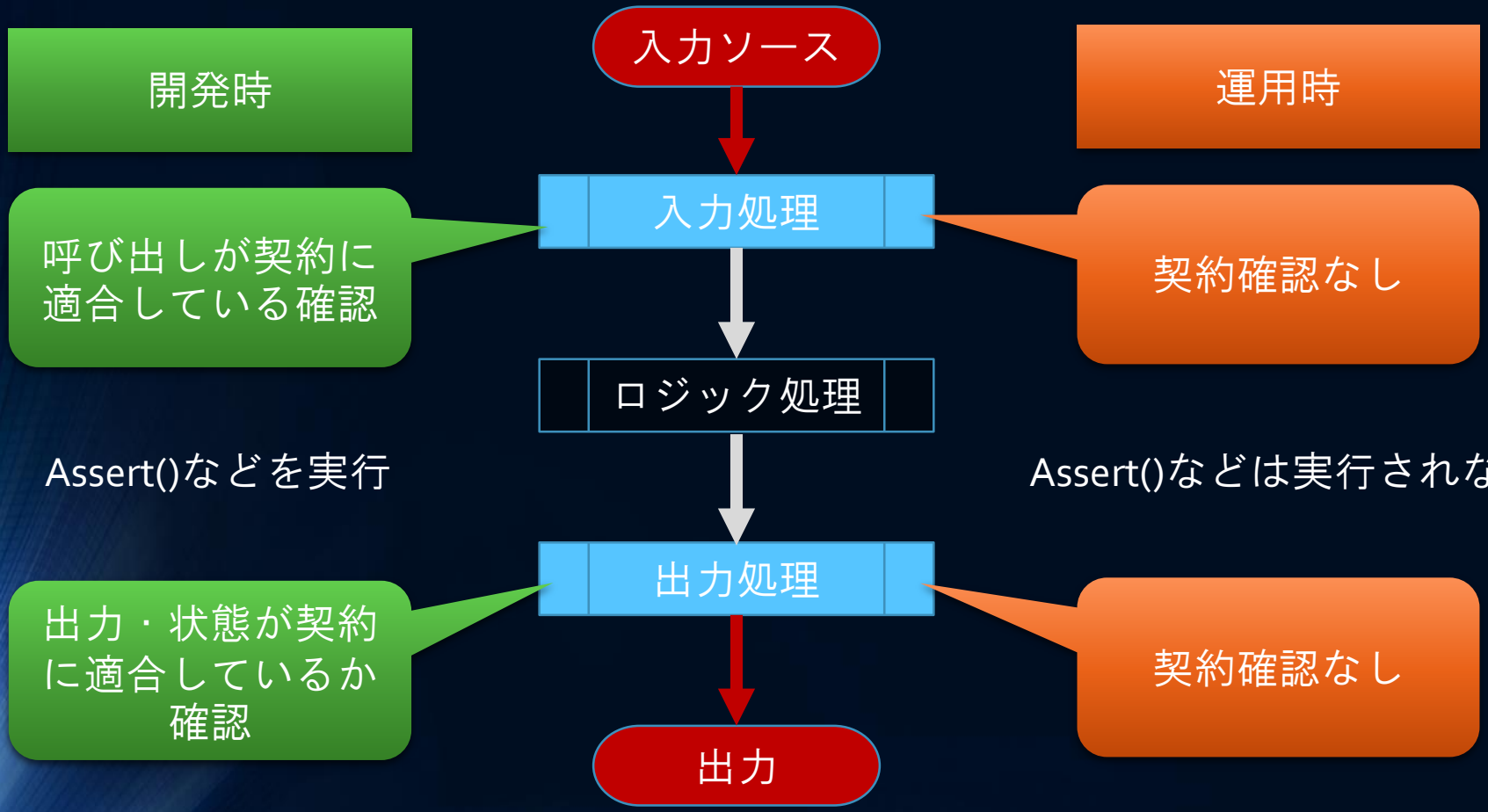
これらは<u>**テキストI/Fでバリデーション・エスケープは必須**</u>

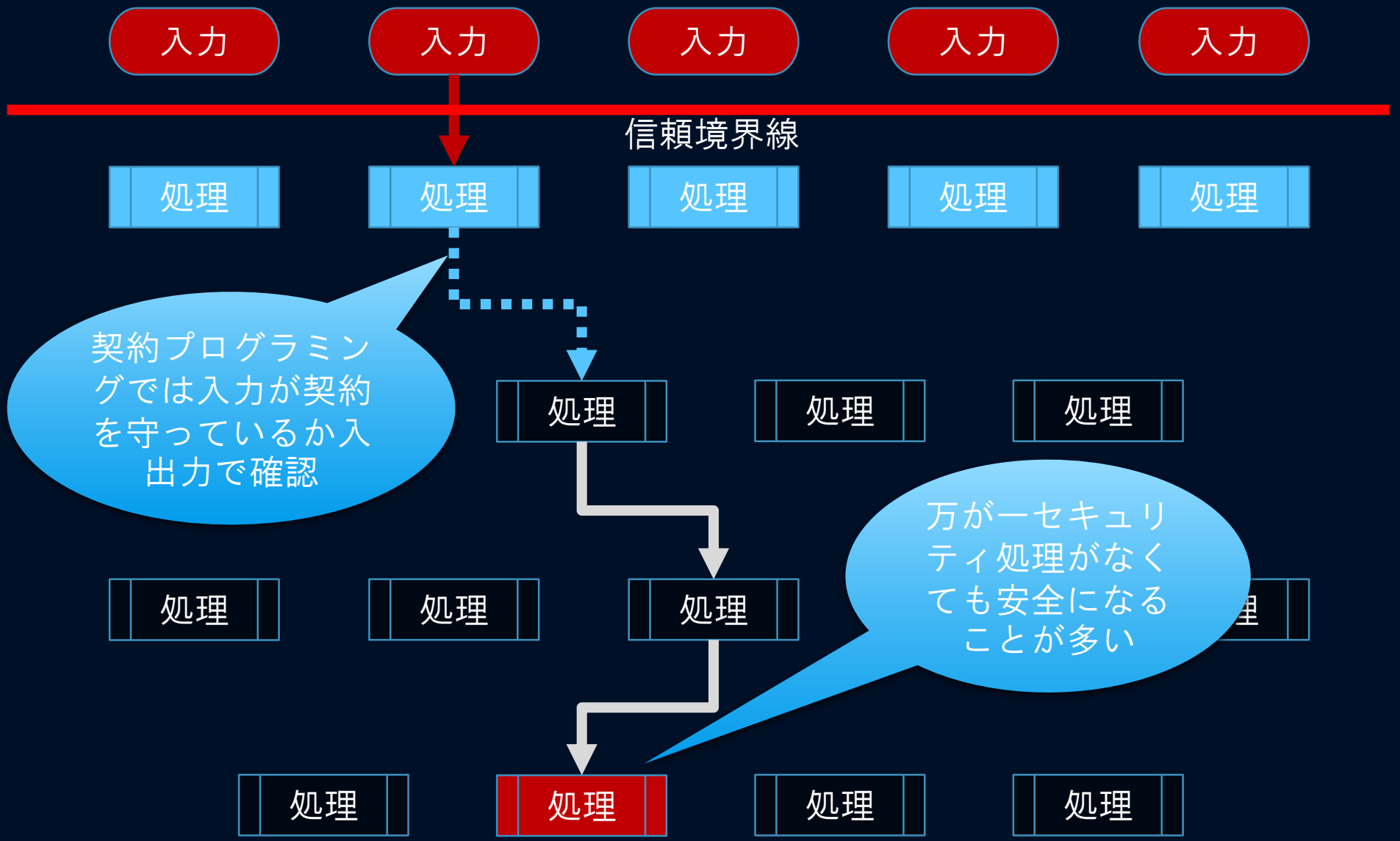




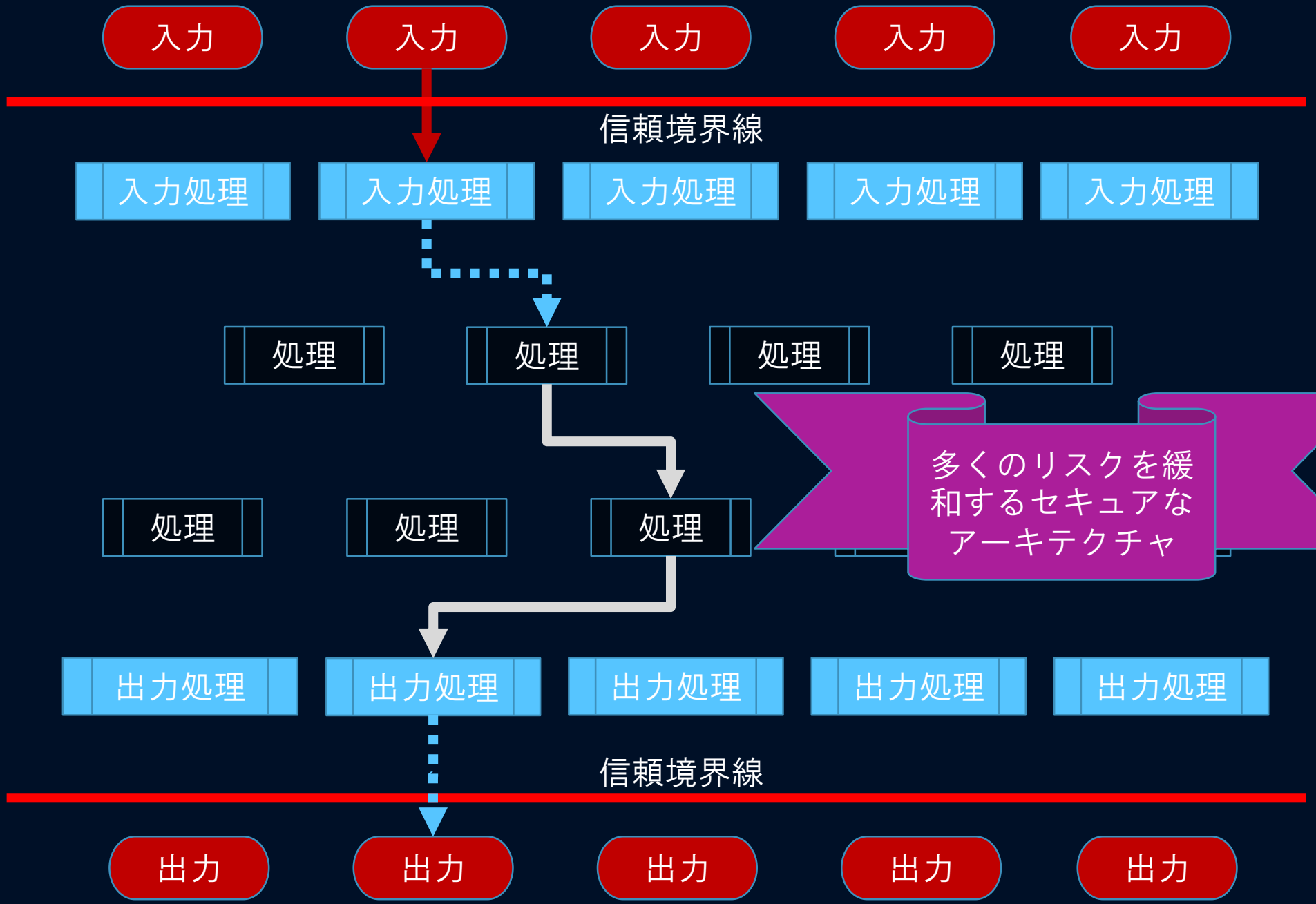
契約プログラミング
契約のよる設計の構造

契約プログラミングの基本





ここで追加のセキュリティ確認も勿論OK。縦深防御（多重のセキュリティ・多層防御）

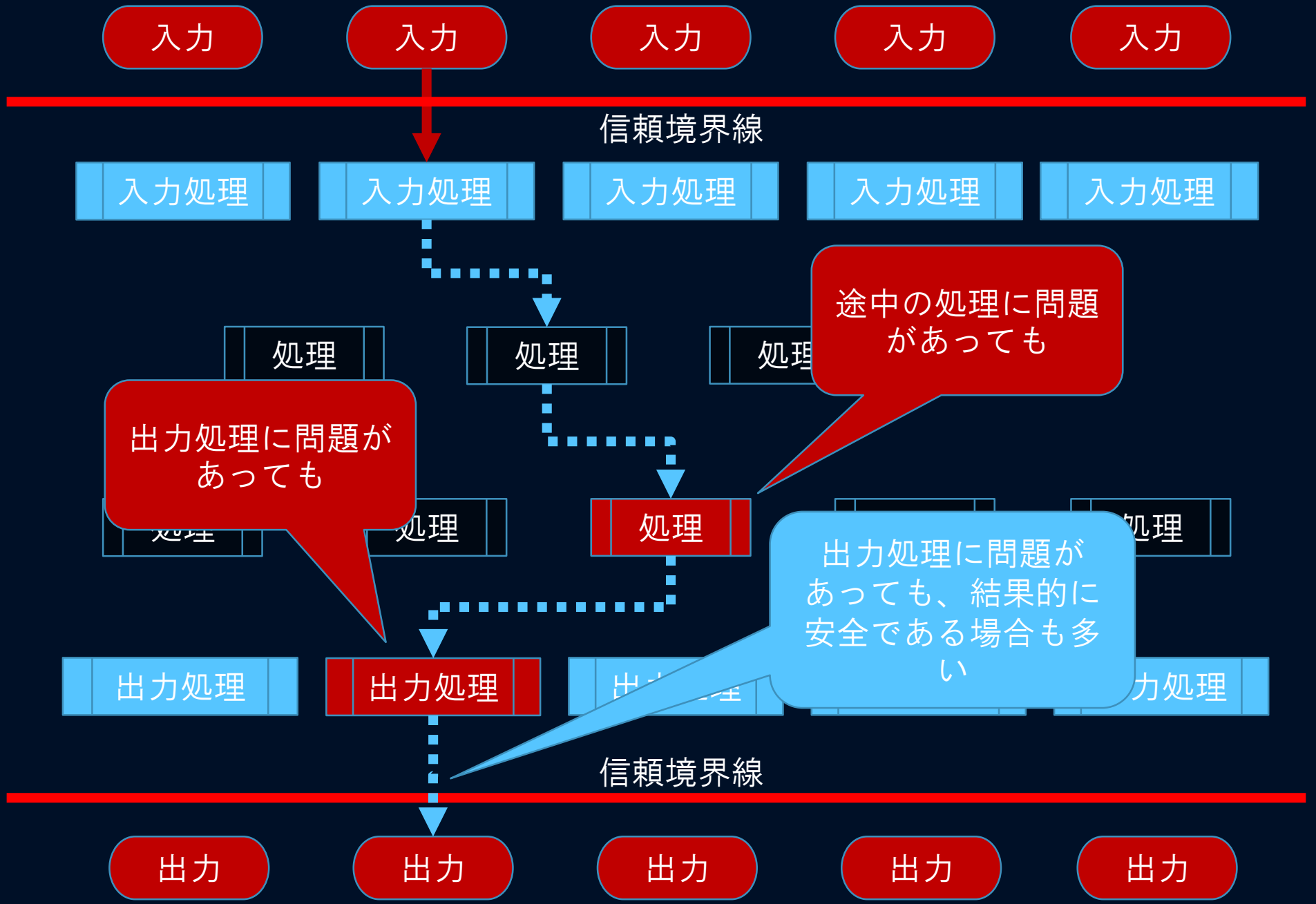

入力
入力
入力
入力
入力
信頼境界線
入力処理
入力処理
入力処理
入力処理
入力処理
処理
処理
処理
処理
処理
処理
処理
多くのリスクを緩和するセキュアなアーキテクチャ
出力処理
出力処理
出力処理
出力処理
出力処理
信頼境界線
出力
出力
出力
出力
出力

セキュアな入出力制御で対策・緩和できる脆弱性

入力	出力	CWE
High		CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
Mod	High	CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
Mod	High	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Mod	High	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Mod		CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
Mod		CWE-131: Incorrect Calculation of Buffer Size
High		CWE-134: Uncontrolled Format String
Mod		CWE-190: Integer Overflow or Wraparound
Mod		CWE-434: Unrestricted Upload of File with Dangerous Type
Mod	Mod	CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
Mod	High	CWE-676: Use of Potentially Dangerous Function
	Ltd	CWE-732: Incorrect Permission Assignment for Critical Resource
Mod		CWE-807: Reliance on Untrusted Inputs in a Security Decision
High		CWE-829: Inclusion of Functionality from Untrusted Control Sphere
DiD		CWE-862: Missing Authorization

「入力」「出力」を確実にするだけで8割9割の脆弱性を防止・緩和可能

職人技セキュリティから
エンジニアリングされた
セキュリティへ


入力
入力
入力
入力
入力
信頼境界線
入力処理
入力処理
入力処理
入力処理
入力処理
処理
処理
処理
処理
処理
出力処理に問題があっても
途中の処理に問題があっても
出力処理に問題があっても、結果的に安全である場合も多い
出力処理
出力処理
信頼境界線
出力
出力
出力
出力
出力

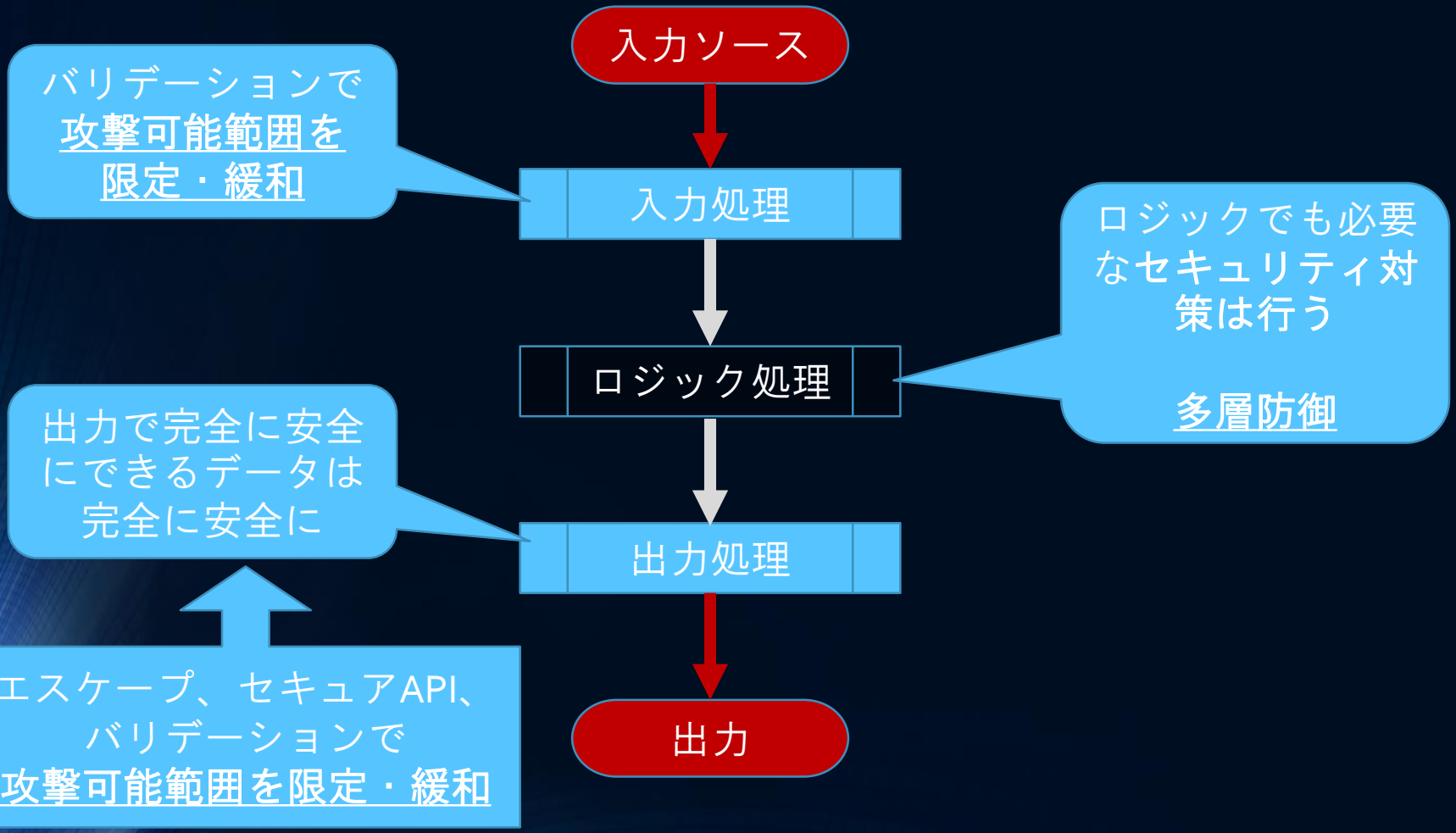
アプリの入力・出力の安全性確保はセキュアアプリの土台・基礎

継続的インテグレーション
高速な開発サイクルも
契約プログラミングで
効率化

間違えてはならないのは
契約プログラミング
契約による設計でも

多層防御は必要！

セキュアなプログラムの基本構造

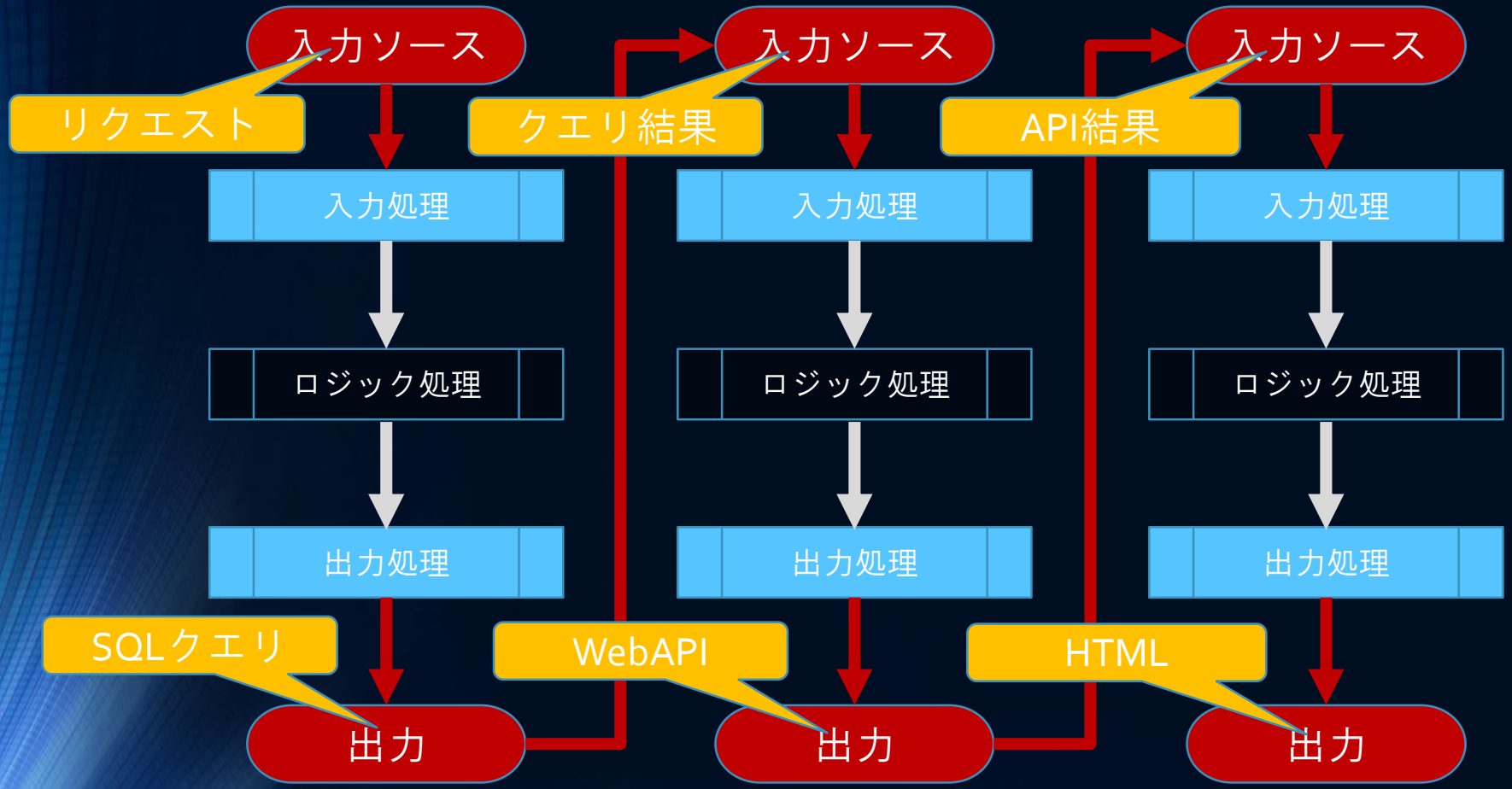


普通のアプリは単純な

「入力→処理→出力」

ではない

セキュアなプログラムの基本構造

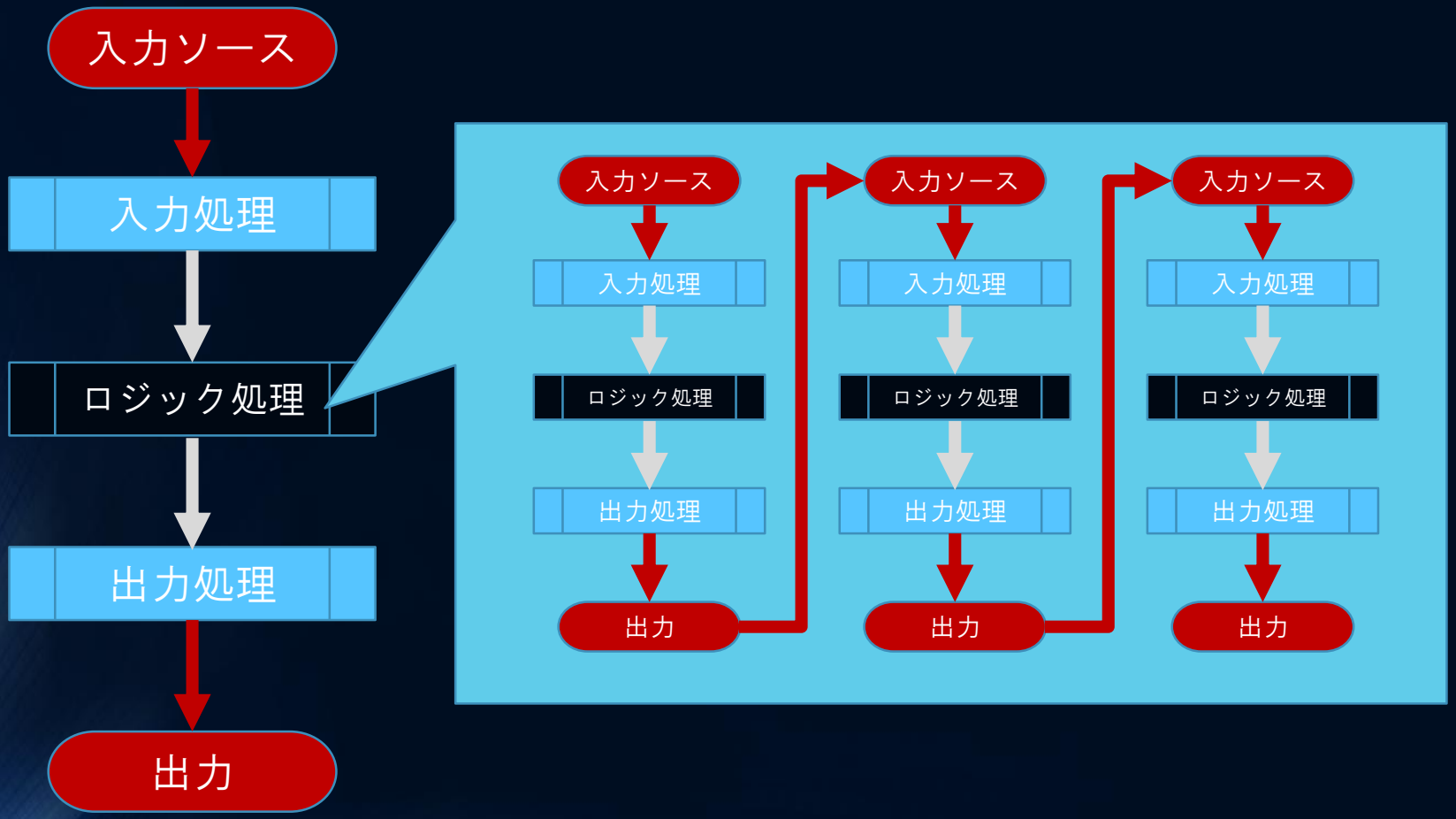


アプリは複数の

「入力→処理→出力」

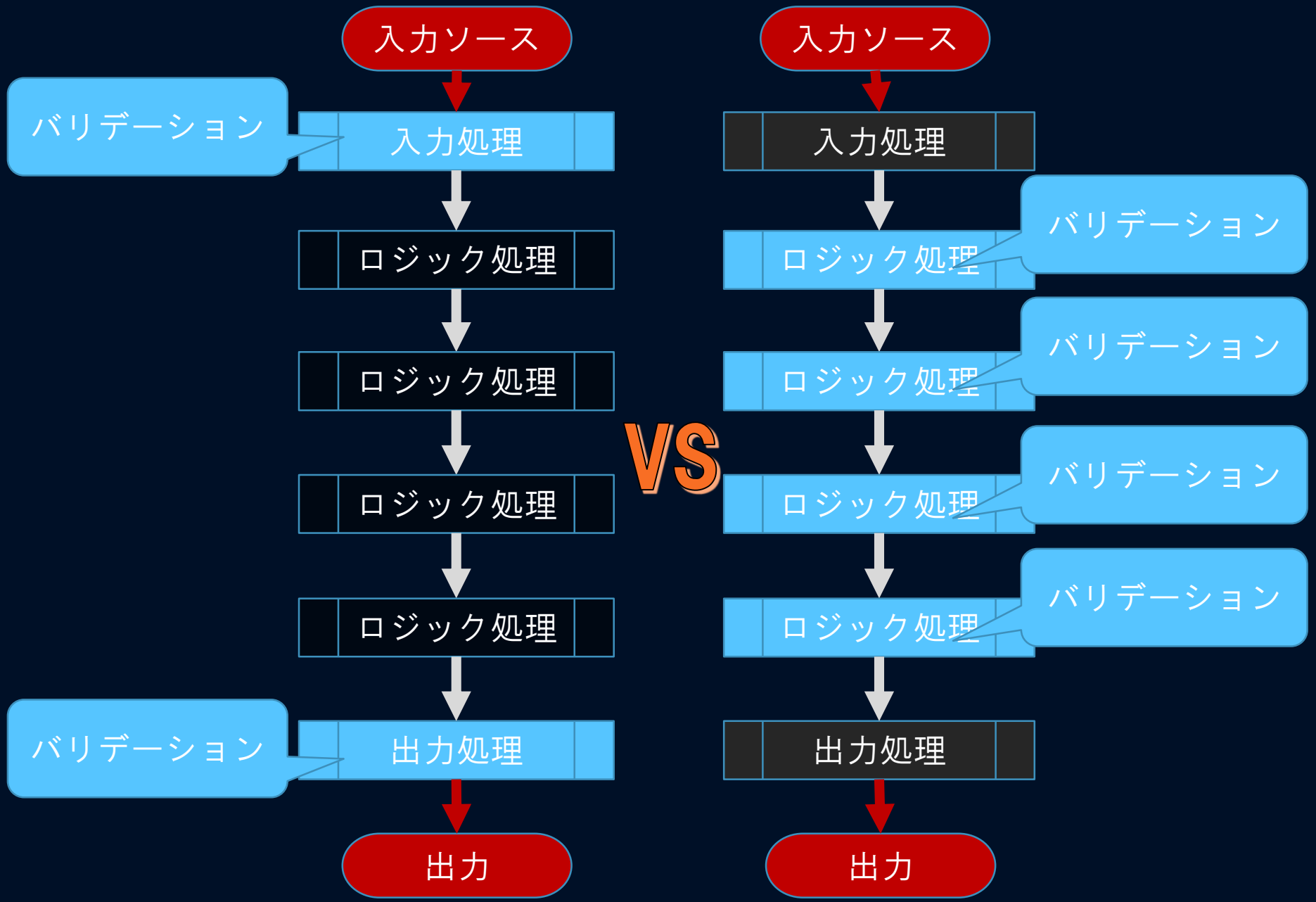
の組み合わせの構造

セキュアなプログラムの基本構造

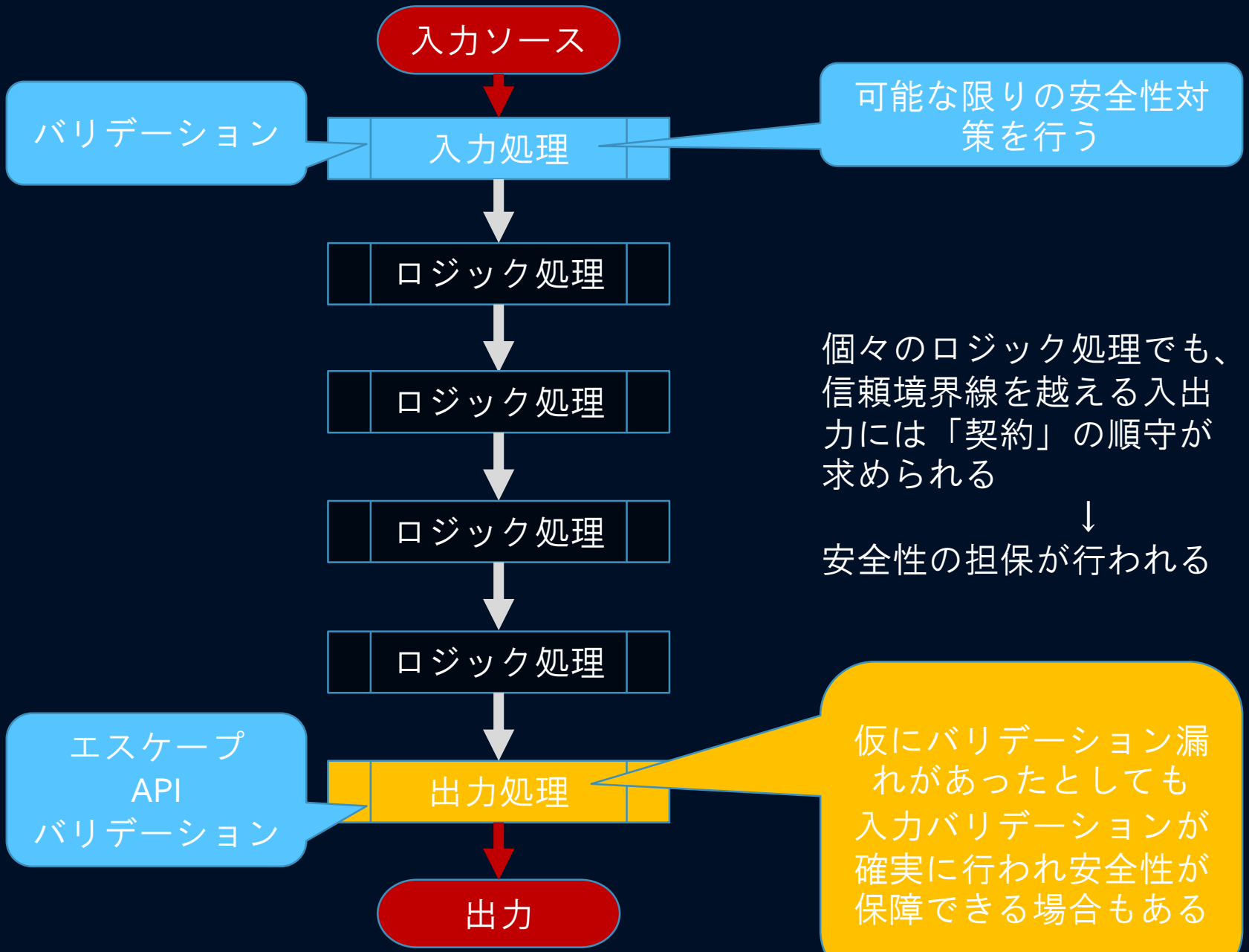


契約プログラミング
契約による設計は
高性能

例：文字エンコーディングのバリデーション


入力ソース
バリデーション
入力処理
ロジック処理
ロジック処理
ロジック処理
ロジック処理
バリデーション
出力処理
出力
VS
入力ソース
入力処理
ロジック処理
ロジック処理
ロジック処理
ロジック処理
バリデーション
バリデーション
バリデーション
バリデーション
出力処理
出力

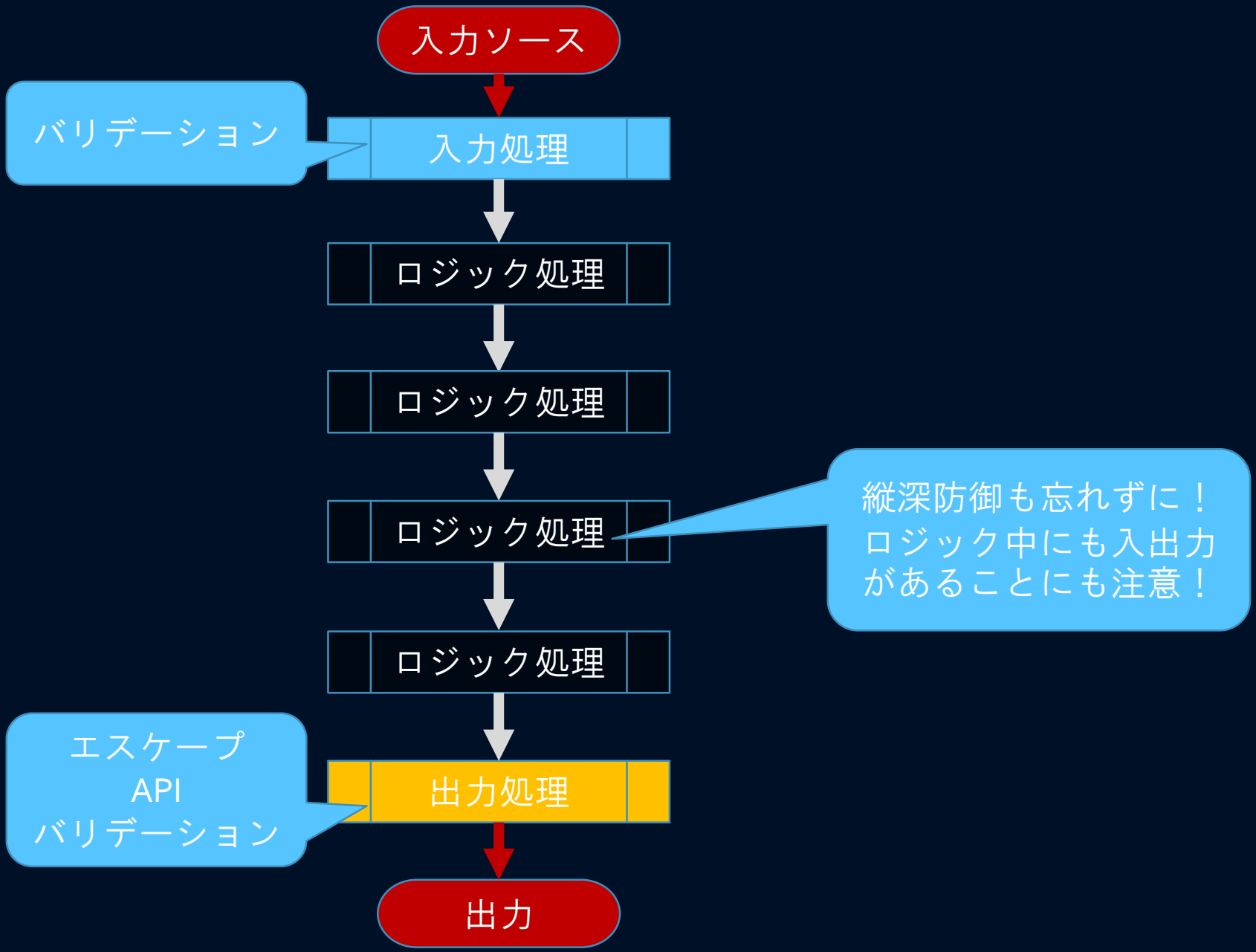
契約プログラミング
契約による設計は
セキュア



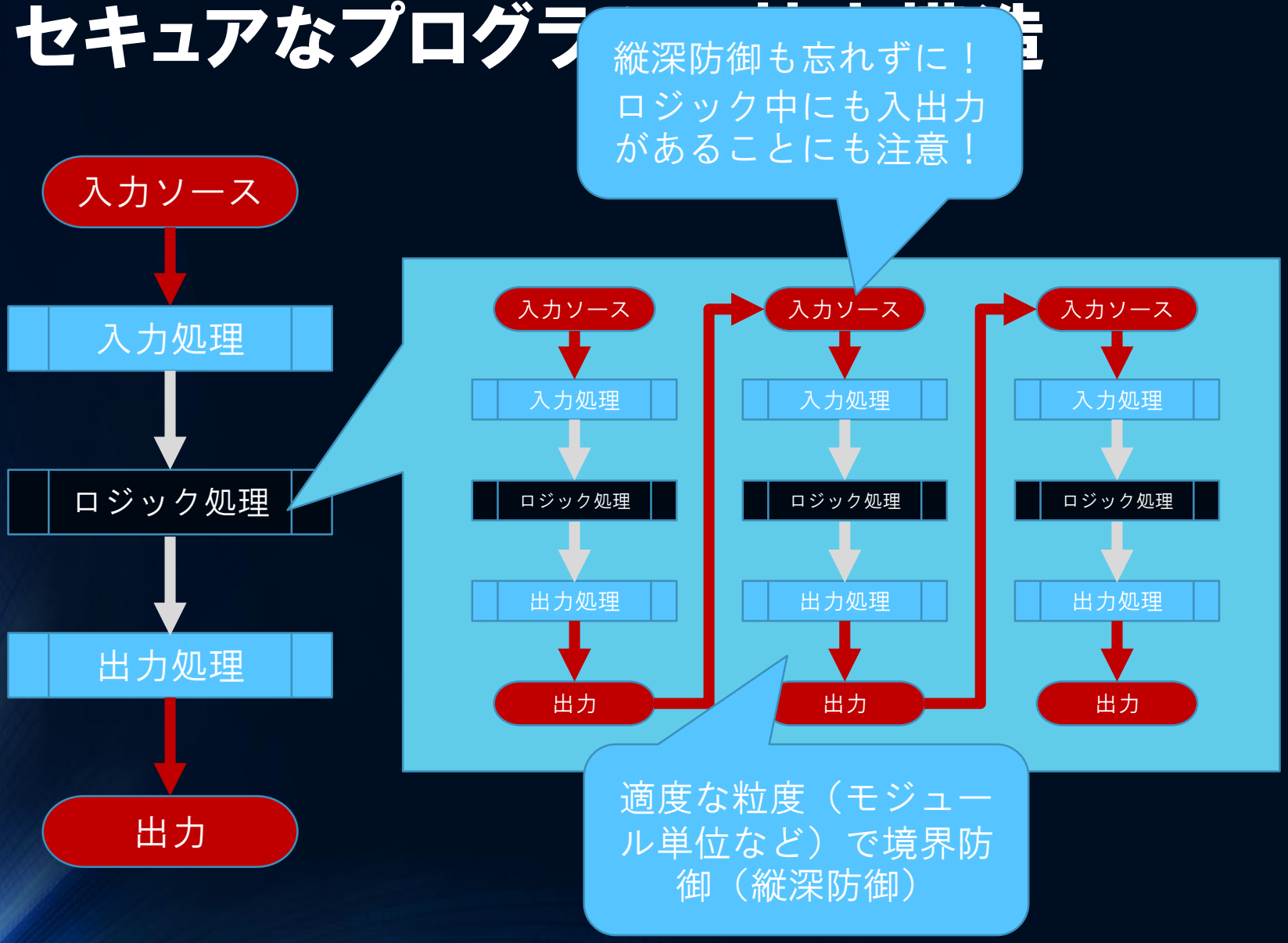
個々のロジック処理でも、信頼境界線を越える入出力には「契約」の順守が求められる

↓

安全性の担保が行われる


入力ソース
バリデーション
入力処理
ロジック処理
ロジック処理
ロジック処理
ロジック処理
縦深防御も忘れずに！
ロジック中にも入出力
があることにも注意！
エスケープ
API
バリデーション
出力処理
出力

セキュアなプログラ


縦深防御も忘れずに！
ロジック中にも入出力があることにも注意！
入力ソース
入力処理
ロジック処理
出力処理
出力
入力ソース
入力処理
ロジック処理
出力処理
出力
入力ソース
入力処理
ロジック処理
出力処理
出力
入力ソース
入力処理
ロジック処理
出力処理
出力
適度な粒度（モジュール単位など）で境界防御（縦深防御）

情報セキュリティ対策とは

- 対策・緩和策を積み重ねて、
  データ/プログラムの安全性を管理

- 安全性を管理する、とは
- リスクを管理する、であり
- **許容可能な程度までリスクを緩和する**

- 万が一のインシデント発生時には原因が判るようにする
  - CIAに加え、Accountabilityも重要なセキュリティ要素
  - この他にAuthenticityも重要なセキュリティ要素

参考にすべき セキュアコーディングガイド

- **CERT Secure Coding Standard**
  - http://blog.ohgaki.net/cert-top-10-secure-coding-standard
- **CWE/SANS TOP 25**
  - http://blog.ohgaki.net/sans-cwe-top-25-monster-mitigation
- **OWASP Secure Coding Practices**
  - http://blog.ohgaki.net/owasp-secure-coding-practices-quick-reference-guide

安全かつ高速な
部品・サービスを作るには、
契約プログラミングが有効

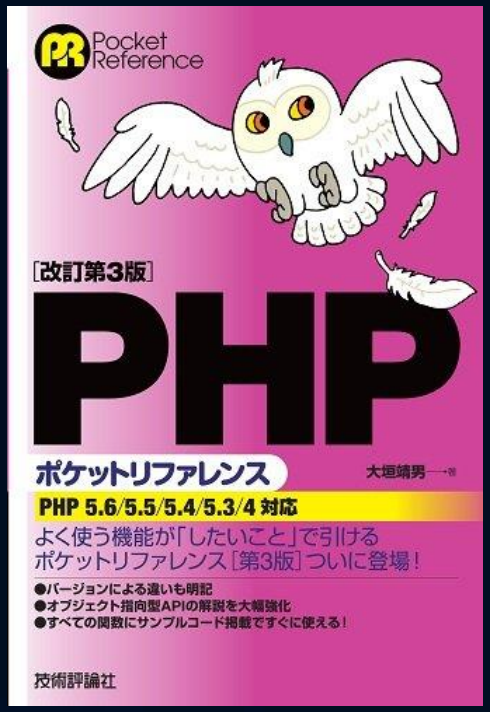
入力バリデーションは
第一のセキュリティ対策
By CERT/SANS/OWASP

入力バリデーションで
未知・うっかり脆弱性にも対応可能

セキュリティ対策として必ず実行

PHPポケットリファレンス 第三版

- カンファレンス会場で販売中



お問い合わせ・ご相談は

Web開発、セキュリティで困ったら

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

ご清聴
ありがとうございました