Rails 4 Security

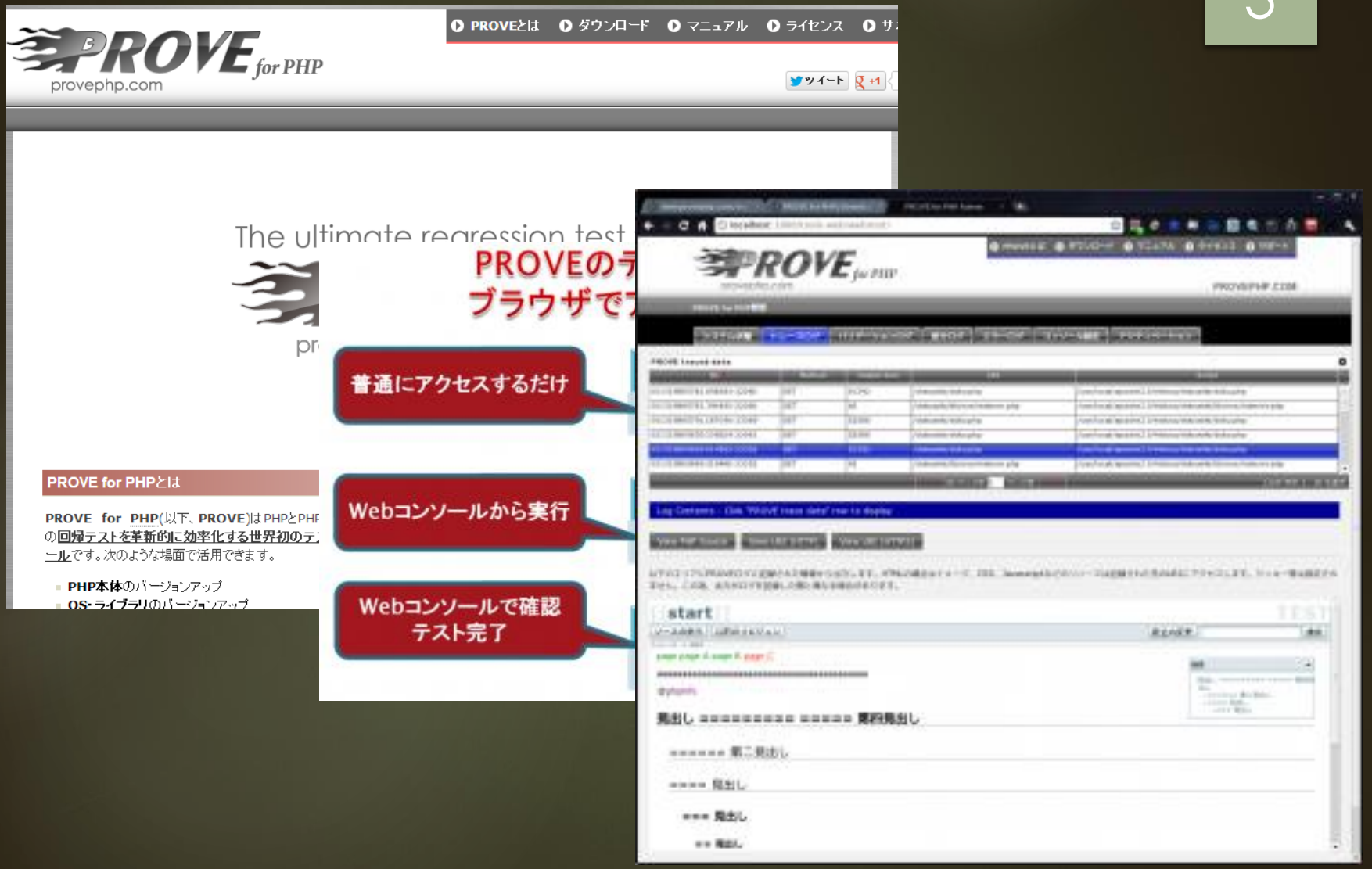
エレクトロニック・サービス・イニシアチブ

HTTP://WWW.E-SI.JP/

自己紹介

- 氏名：大垣靖男（Ohgaki Yasuo）
- メール：yohgaki@ohgaki.net
- SNS:　yohgaki (Twitter/Facebook/Gmail/LinkedIn）
- 職業：
  - エレクトロニック・サービス・イニシアチブ有限会社社長
  - PHP技術者認定機構顧問・BOSS CON CTO・岡山大学大学院講師
  - Webアプリソースコード検査（Ruby/PHP/Java/C#/ObjectiveC）など
  - OSS開発者：PHP･Momonga-Linuxコミッター

PROVE for PHP



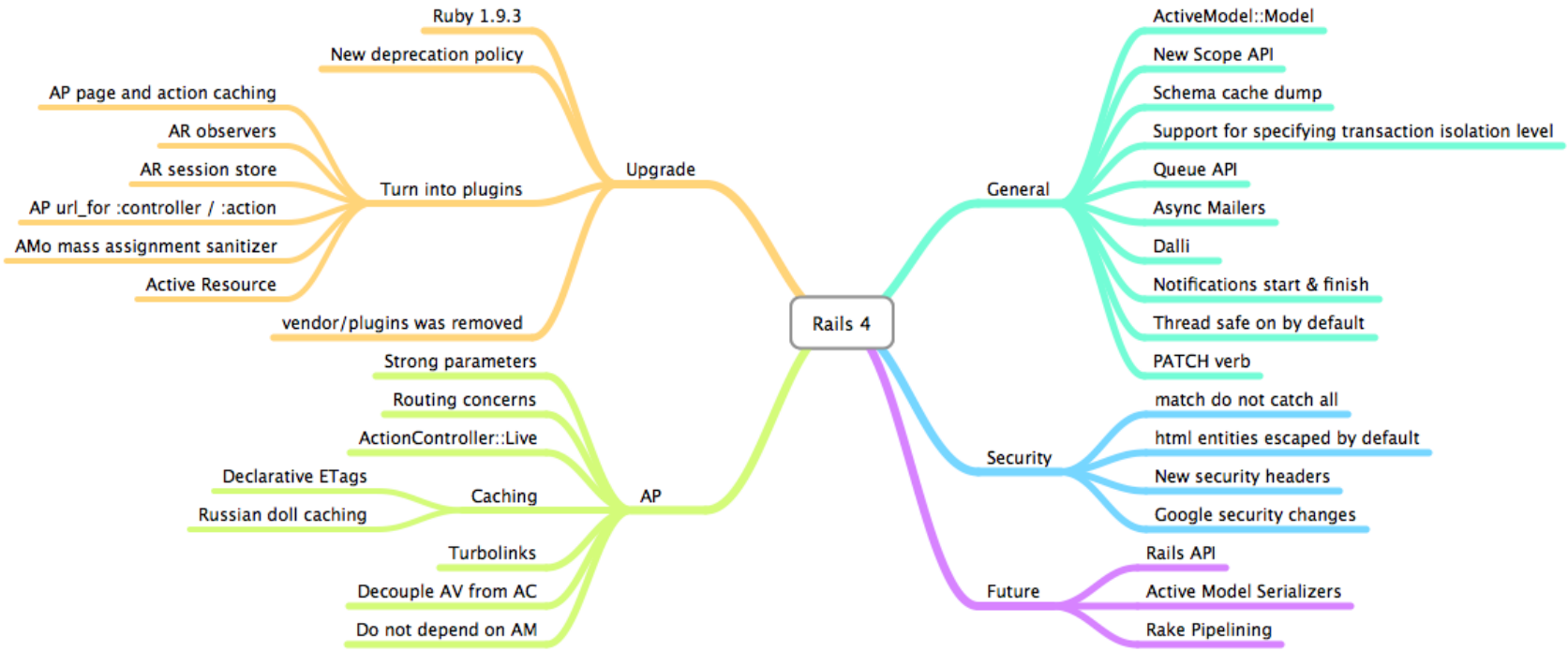
Windows7でRails4

注意：
すべてのセキュリティ対策を解説していません！

このスライドに記載されている対策のみが対策ではありません。

Rails4

Rails4



http://edgeguides.rubyonrails.org/4_0_release_notes.html

Rails4 Major Features

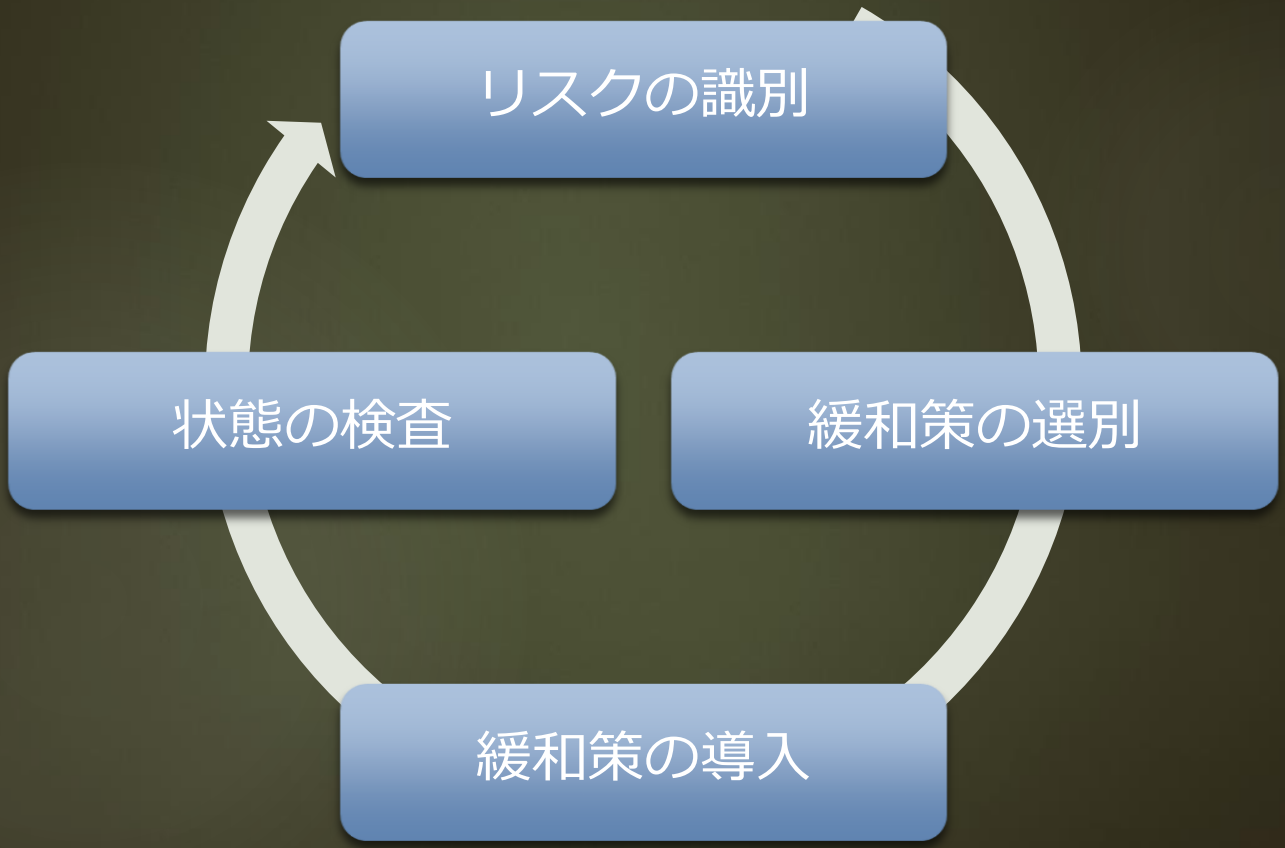
- Ruby 2.0推奨、1.9.3+以上
- Turbolinks
- Russiaon Doll Caching
- Strong Parameters

http://edgeguides.rubyonrails.org/4_0_release_notes.html

ITセキュリティ

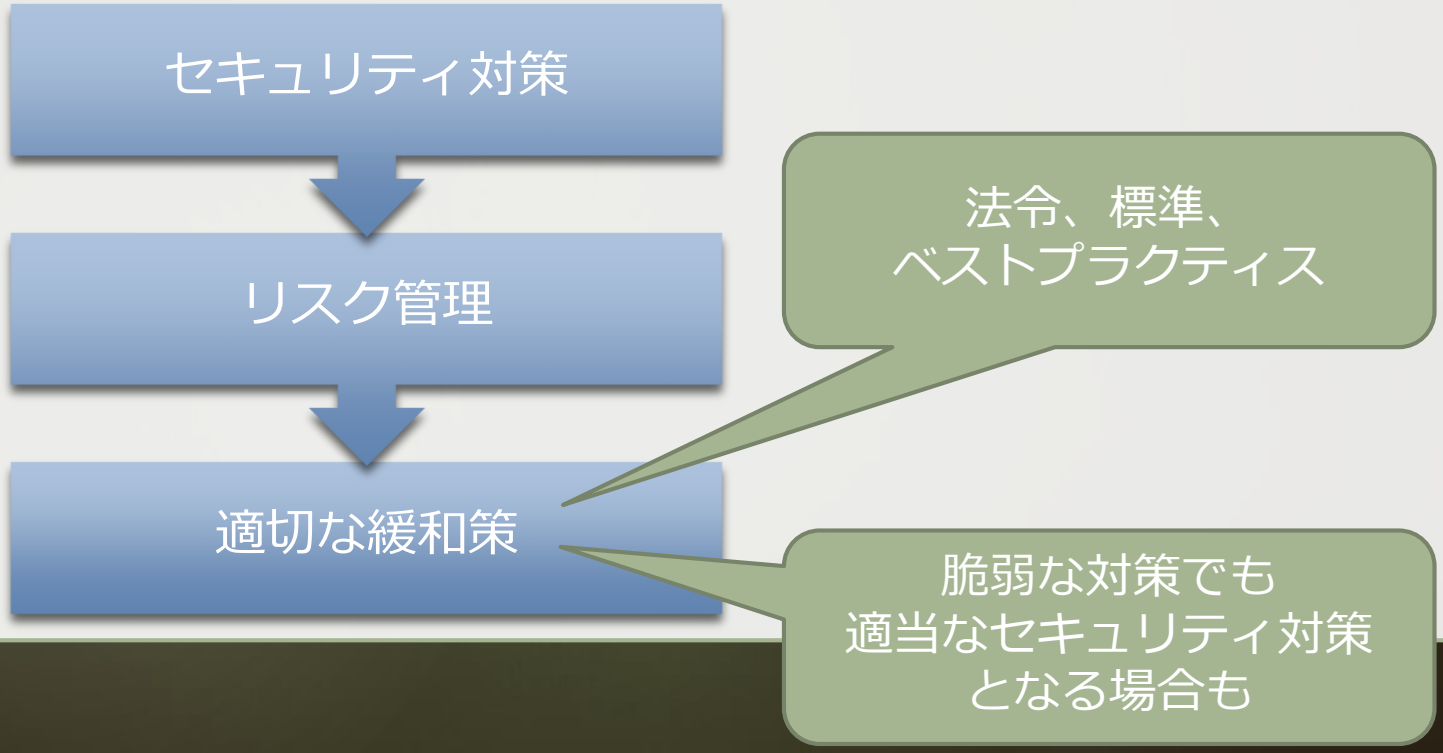
ITセキュリティ？

ITセキュリティ対策？



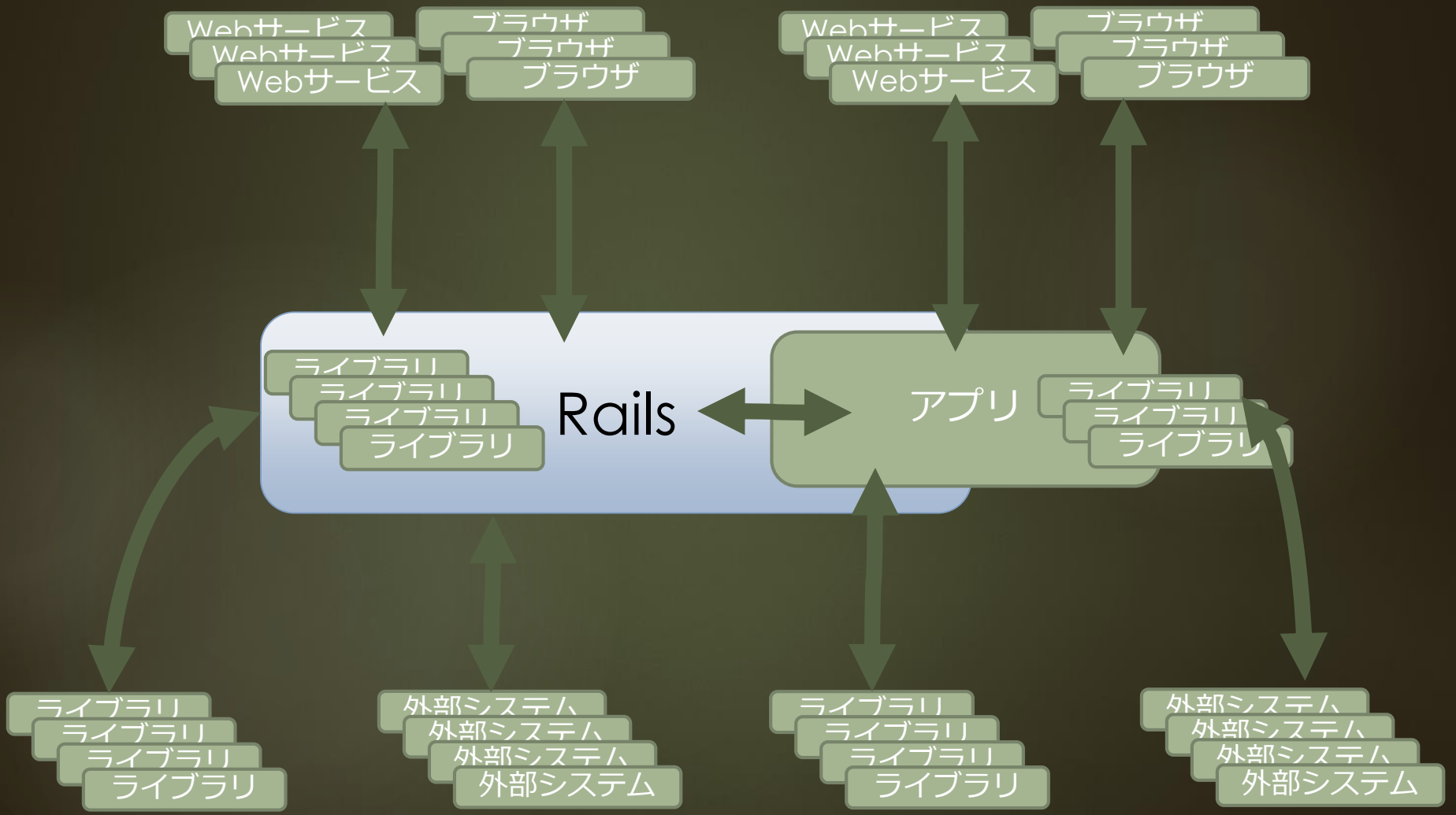
よくあるカン違い

セキュリティ対策は完璧でなければならない！

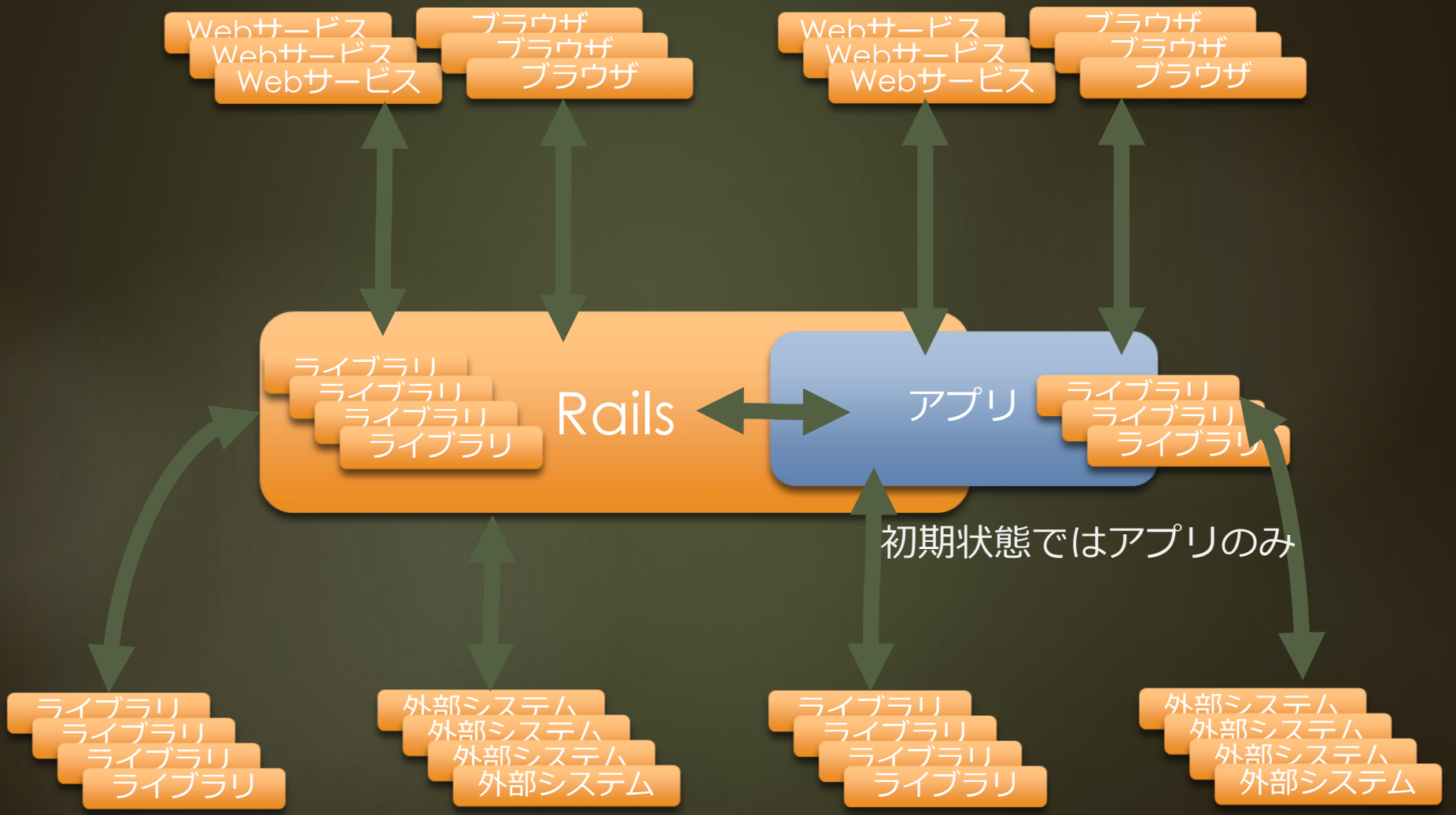


構造とセキュリティ

アプリの基本構造



信頼できるもの？



プログラムの基本構造

入力
- 入力値のバリデーション
- 入力値により適切なビジネスロジックを選択

脆弱性

処理
- 定義された処理を実行

脆弱性

出力
- 出力先の入力仕様に適合した出力の生成
- エスケープ、ヘルパー、バリデーション

脆弱性

入力バリデーション

ベストプラクティス

- 入力パラメータは入力処理として
  バリデーションする

バッドプラクティス

- 入力パラメータを入力処理ではない部分で
  バリデーションする

セキュリティ対策の基本 入力・出力制御

入力バリデーション

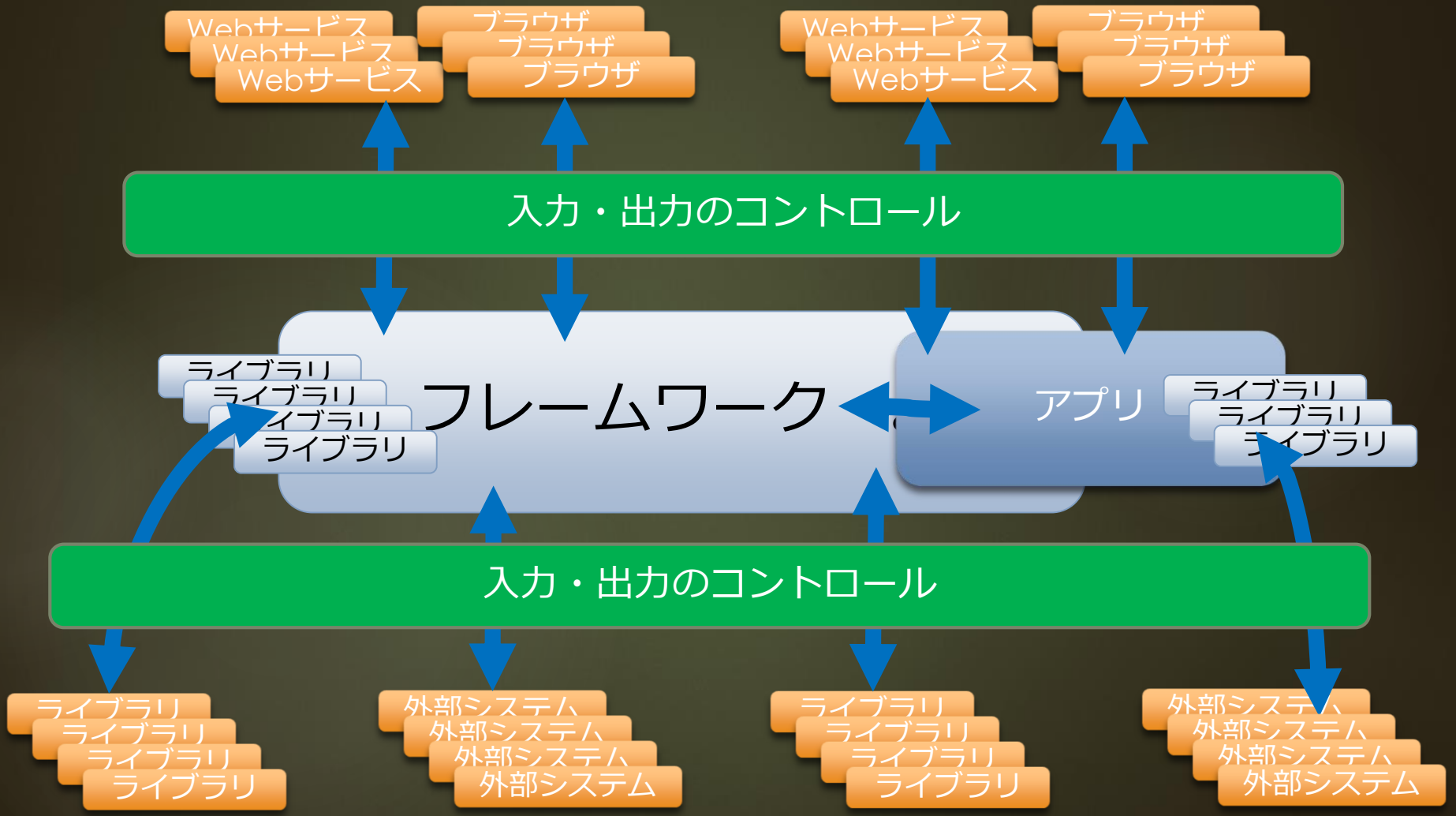
- SANS/CWE TOP25 Monster Mitigation #1
- ISO27000 / ISMS
- 多くのベストプラクティス、セキュリティ標準

出力コントロール

- SANS/CWE TOP25 Monster Mitigation #2
  - **エスケープ → ヘルパー → バリデーション**

順番が重要

入力・出力のコントロール



Strong Parameter

最も重要な変更
Strong Parameter

Controller

入力

- 入力値のバリデーション
- 入力値により適切なビジネスロジックを選択

Rails４のバリデーションは
Controller

Model

処理

- 定義された処理を実行

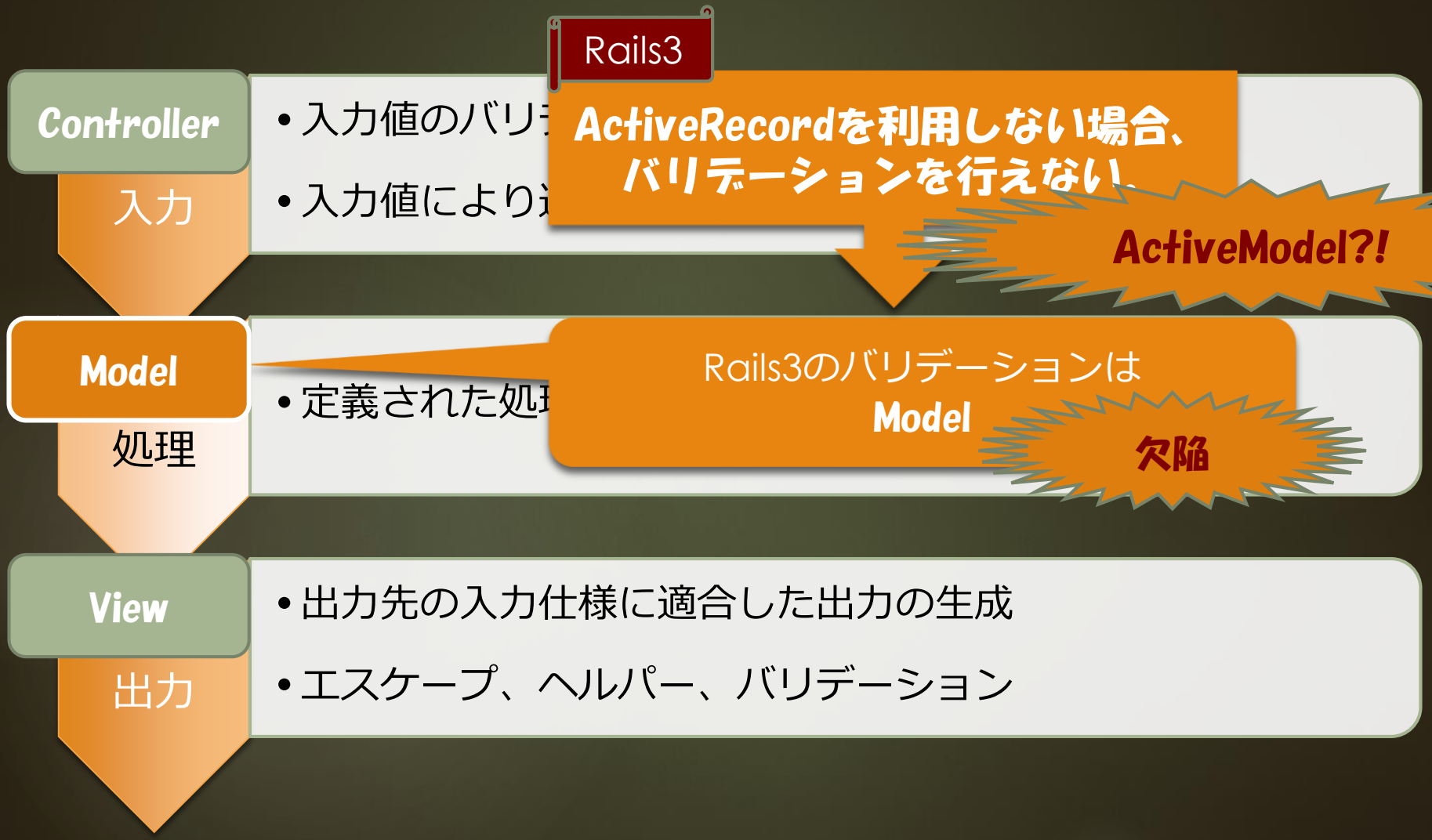
Rails3のバリデーションは
Model

View

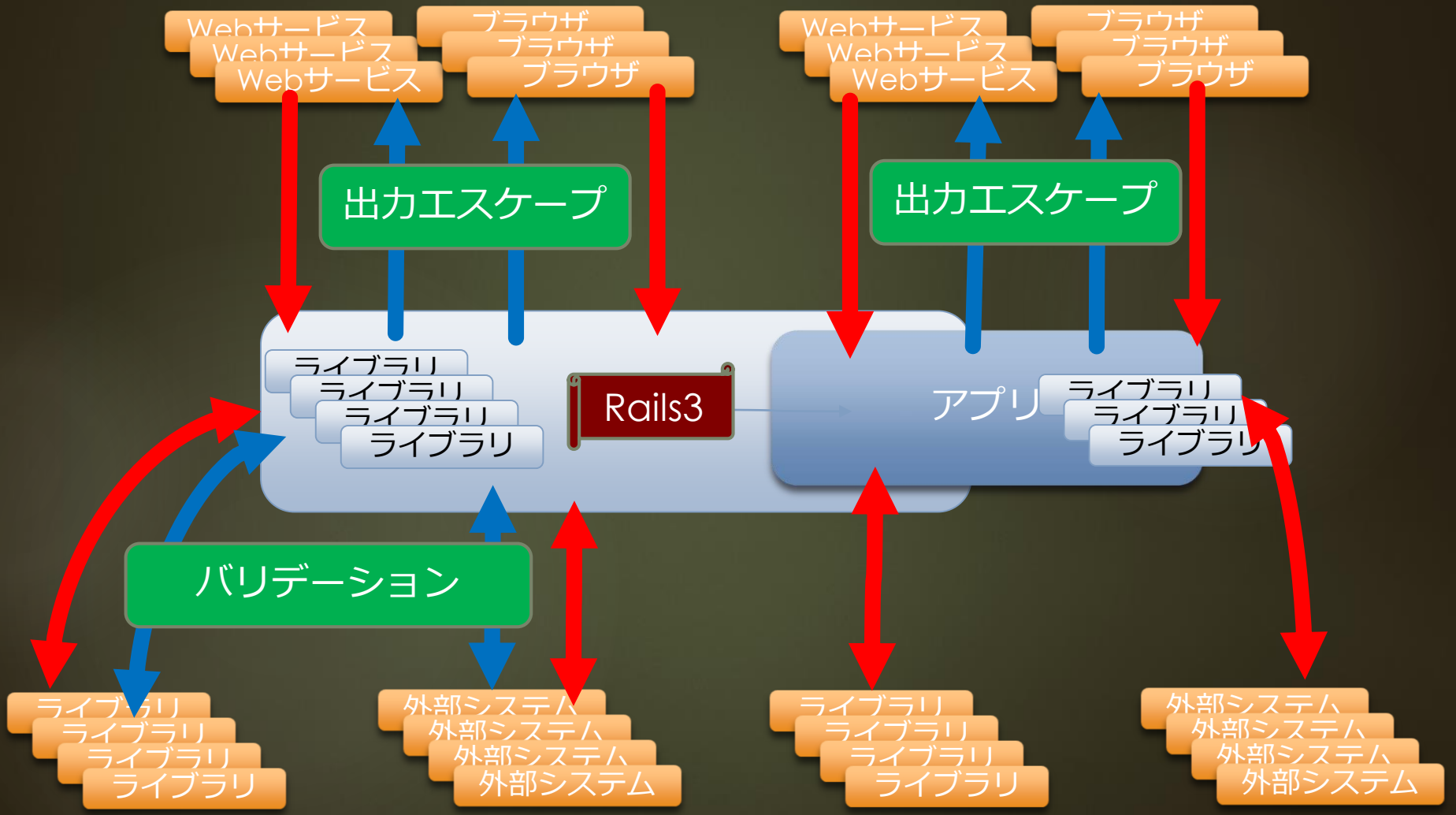
出力

- 出力先の入力仕様に適合した出力の生成
- エスケープ、ヘルパー、バリデーション

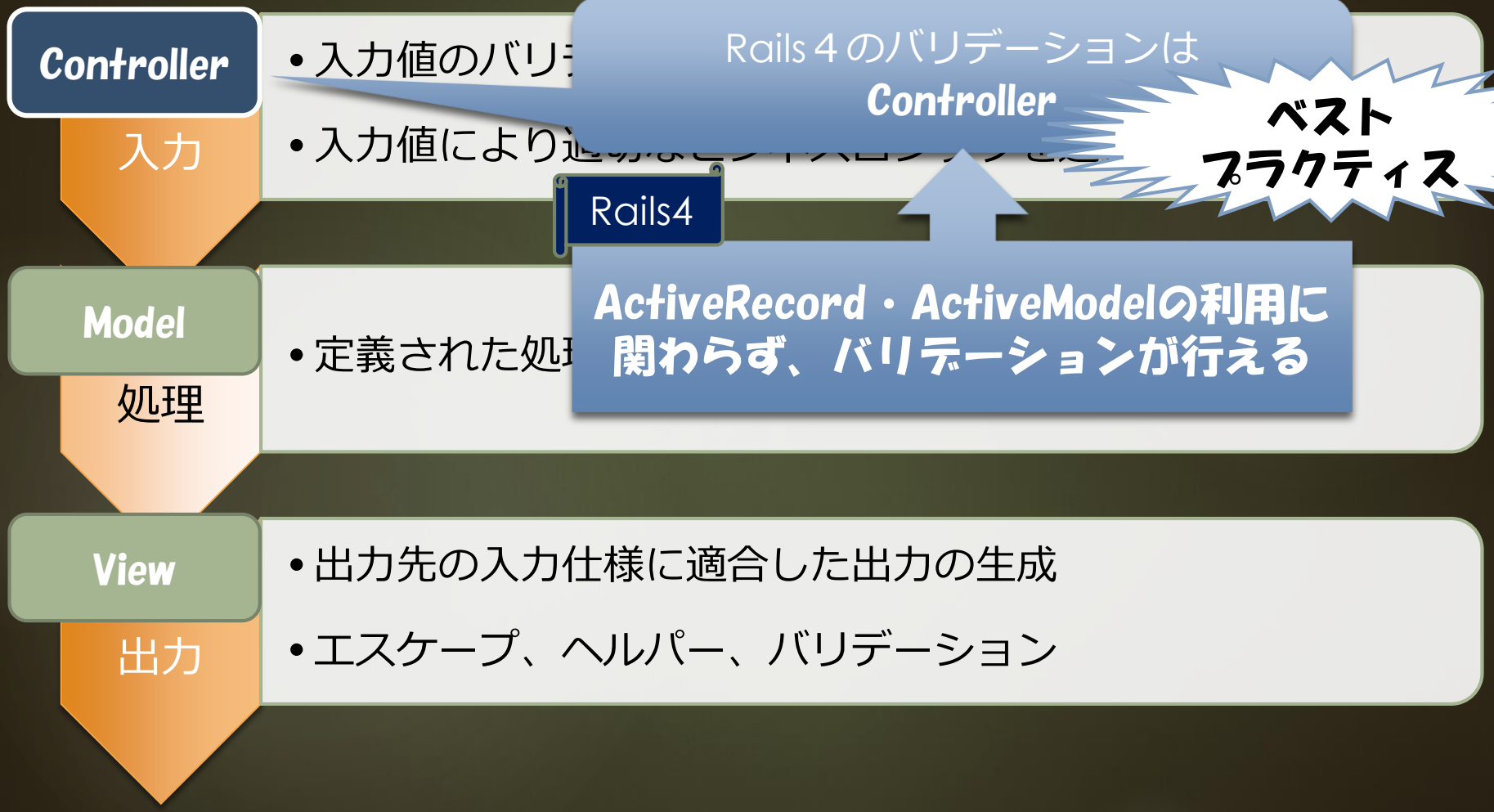
Rails3



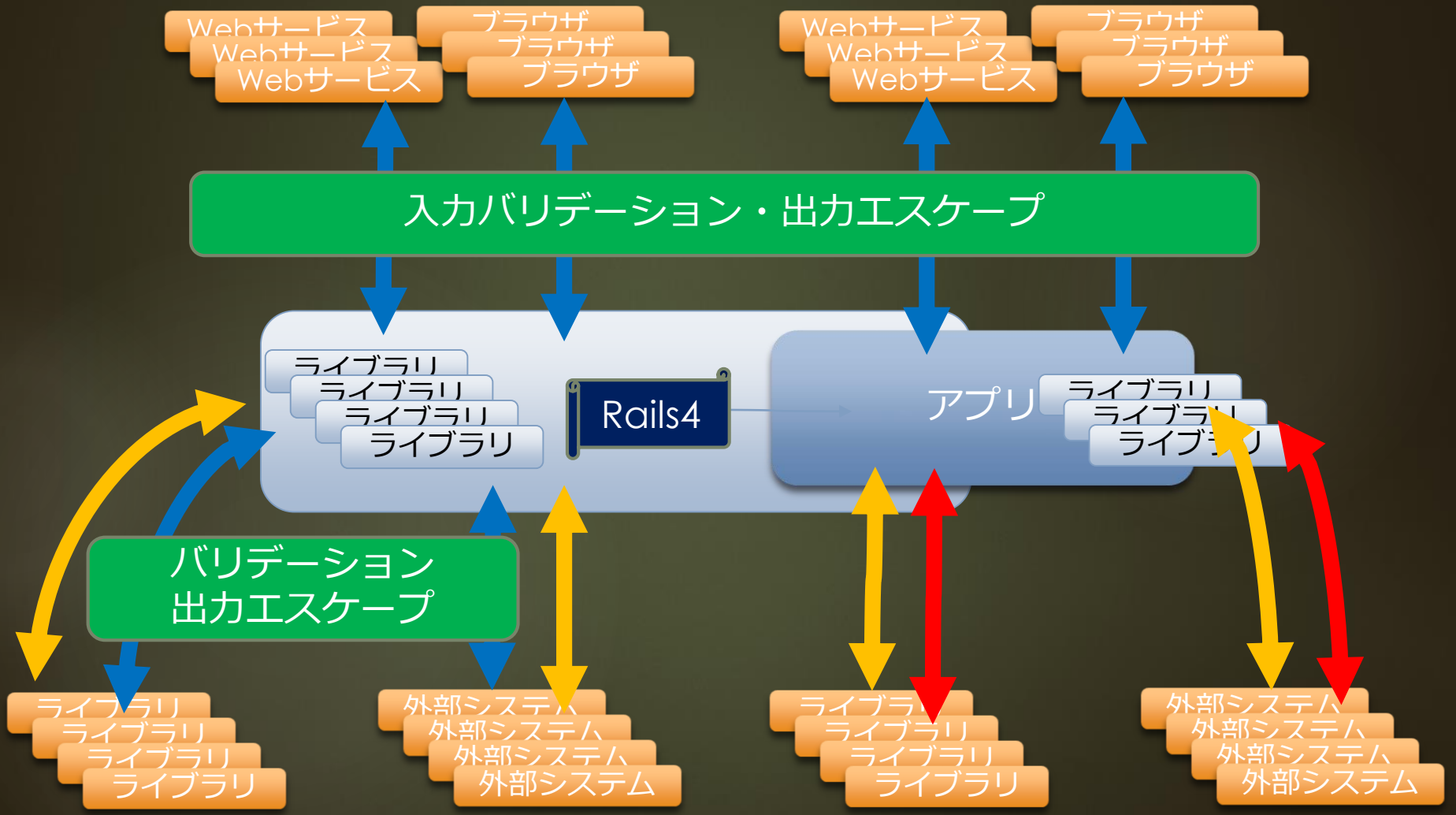
Rail3の入出力制御



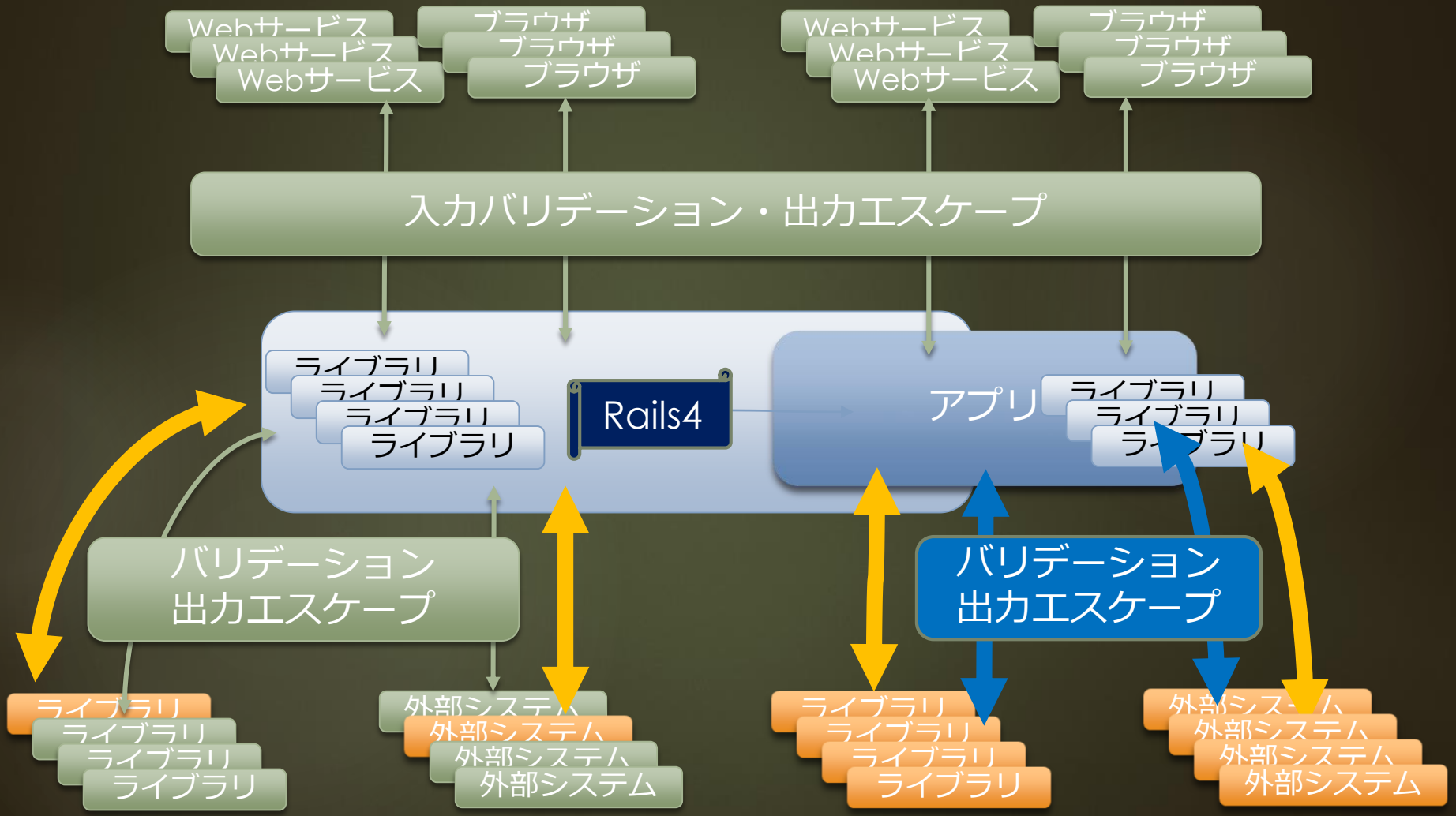
Rails4



Rails4の入出力制御



セキュリティ対策ポイント



古い手法
Weak Parameter

Rails3

```
@post.update_attributes(post_params)
```

Modelのバリデーションが動作

Rails3

```
@post.update_attributes(:user, 'User Name')
@post.update_column(:user, 'User Name')
```

Modelのバリデーション無し

新しい手法
Strong Parameter

Controllerでバリデーションが動作

Rails4

```
class UsersController < ApplicationController
 private
 def user_params
  params.require(:user).permit(:name, :email).validate(self)
 end
end
```

追加するだけ

メソッドチェインでバリデーション

Rails3 https://github.com/rails/strong_parameters

古い手法の変更点

Rails3

```
@post.update_attributes(post_params)
```

Rails4

```
@post.update(post_params)
```

Modelのバリデーションが動作

Rails3

```
@post.update_attributes(:user, 'User Name')
@post.update_column(:user, 'User Name')
```

Rails4

```
@post.update_column(post_params)
```

Modelのバリデーション無し

Modelのバリデーション？

Modelのバリデーションは行うべきか？

- 多重のセキュリティ・フェイルセーフ

ベスト
プラクティス

- Modelでもバリデーションを行う方がより安全
  - ただしStrong Parameterと全く同じバリデーションであればあまり意味がない
- ゆるいホワイトリストバリデーションが適当
  - 例：数値なら文字が数字だけか？文字列なら常識的な長さを超えないか？など

Defense for Major Threats

Javascript インジェクション

ERB：デフォルトHTMLエスケープ

- 直接出力には<%= raw var %>, <%== var %>
  - h, html_escape*, json_escape, sanitize_css
- 無効化は<% somePost.content.html_safe %>
- URLエンコード：url_encode(s)

ActionView::Helpers::*

- FormHelper: check_box, etc
- FormTagHelper: HTMLフォームタグ
- UrlHelper: link_to*, button_to, mail_to
- SanitizeHelper: sanitize*, strip_*
- JavaScriptHelper: escape_javascript
- その他、沢山

Javascript インジェクション

Javascriptインジェクションは最も厄介な問題

- 多岐に渡る攻撃ソース
  - フォーム、URL、クッキー、JSON、データベース、ログファイル、ヘッダー、メール、etc
- ブラウザは「**複数**」の出力先
  - HTTPヘッダー、HTMLコンテント、URL、CSS、Javascript、JSON
- DOMベースJavascriptインジェクション
  - クライアント側でのインジェクションが可能
  - クライアントで実行されると全くログに残らない（検出不可能）

総合的な対策が重要

- 学習とトレーニング – さまざまな原因と攻撃手法と対策を知る

SQLインジェクション

ActiveRecord

- ORMにより保護 – find, update, etc
- プレイスホルダ

Rails4

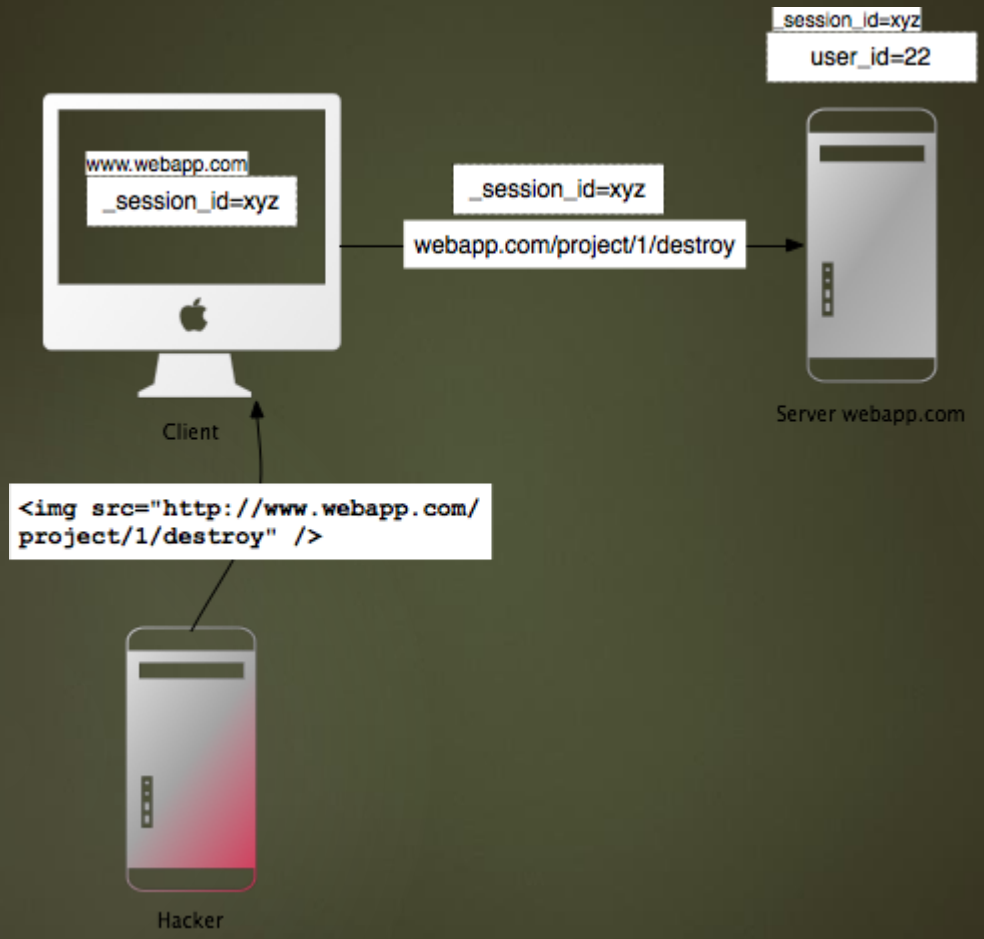
```
User.where(["name = ? and email = ?", "Joe", "joe@example.com"])
# SELECT * FROM users WHERE name = 'Joe' AND email = 'joe@example.com';
User.where(["name = :name and email = :email",
                    { name: "Joe", email: "joe@example.com" }])
# SELECT * FROM users WHERE name = 'Joe' AND email = 'joe@example.com';
```

ActiveRecord::Sanitization

- sanitize*メソッドでエスケープ

```
sanitize_sql(["name='%s' and group_id='%s'", "foo'bar", 4])
                    → "name='foo''bar' and group_id='4'"
```

CSRF



http://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf

CSRF

ビルトイン

- ActionView

```
<head> <%= csrf_meta_tags %> </head>
```

```
# File actionpack/lib/action_view/helpers/csrf_helper.rb, line 19
def csrf_meta_tags
 if protect_against_forgery?
  [
    tag('meta', :name => 'csrf-param', :content => request_forgery_protectio
    tag('meta', :name => 'csrf-token', :content => form_authenticity_token)
  ].join("¥n").html_safe
 end
end
```

- Controller

```
protect_from_forgery secret: "12345678901234567890123456789O..."
```


CSRF

CSRF対策のオプション

Rails4のデフォルト（推奨）

Rails4

```
protect_from_forgery with: :exception # default
protect_from_forgery with: :null_session # セッションを空に
protect_from_forgery with: :rest_session # 新しいセッション
```

Rails3のデフォルト

Session管理

SessionIDの乗っ取り

- 攻撃手法：Javascriptインジェクション、盗聴
- 盗聴対策にはSSL有効化が効果的
  - config.force_ssl = true

Session IDはMD5

- コリージョンディテクション（？）
- 特に問題なし

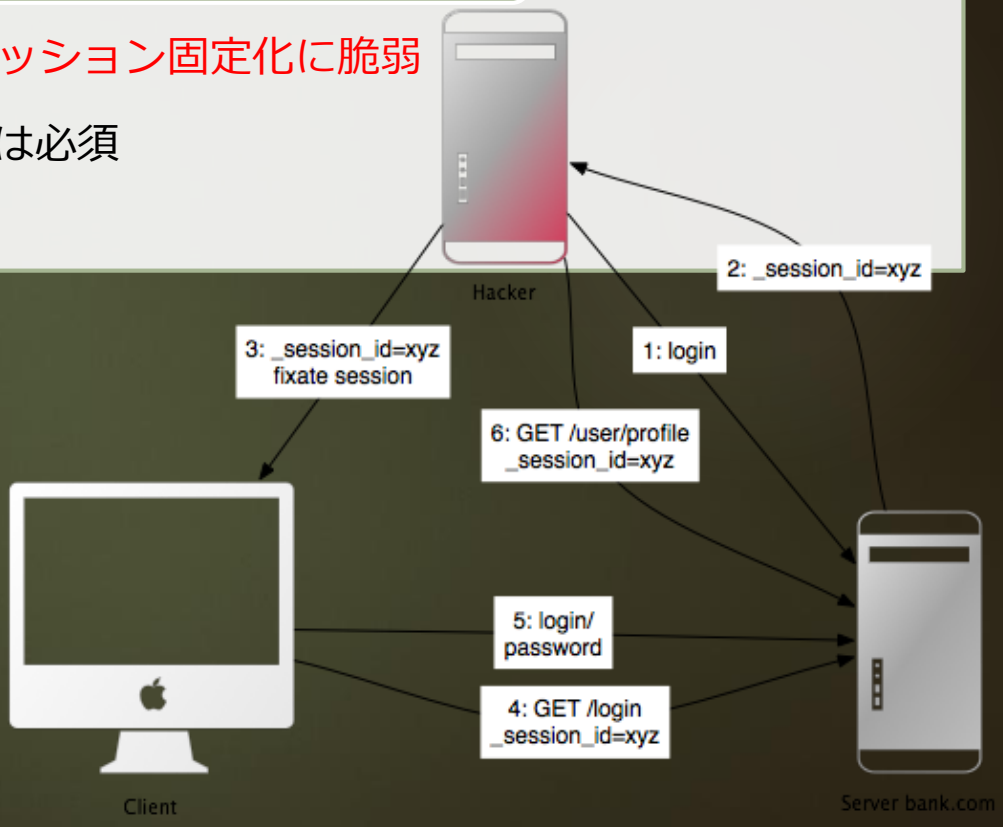
SessionのデフォルトはCookieStore

- ！ベストプラクティス
- Rails4では暗号化　CWE-311 SNS/CWE TOP 25 #8

Session Fixation

セッションIDの固定化

- Railsのセッション管理機構もセッション固定化に脆弱
- ログイン後のセッションID更新は必須
  - reset_session



http://guides.rubyonrails.org/security.html#session-fixation

ファイル
アップロード・ダウンロード

ディレクトリトラバーサル

- ファイル名に"../"を付けて他のディレクトリに保存
- ファイルアップロードに限らずファイル名を使う場合に注意

Railsセキュリティガイドの例

```
def sanitize_filename(filename)
 filename.strip.tap do |name|
  # NOTE: File.basename doesn't work right with Windows paths on Unix
  # get only the filename, not the whole path
  name.sub! /¥A.*(¥¥|¥/)/, ''
  # Finally, replace all non alphanumeric, underscore
  # or periods with underscore name.gsub! /[^¥w¥.¥-]/, '_'
 end
end
```

問題

- "../"を削除するだけで十分か？不十分か？

ブラックリスティング 簡単な物でも間違えやすい

''../''を削除するだけで十分か？不十分か？

- ディレクトリトラバーサル攻撃でよくある間違い

問題 1

- Unicode正規化やエンコーディング

問題 2

- "../''を削除するだけでは簡単に攻撃可能

```
# 入力文字列：  .../.../
# 処理：          ../ を削除
# 出力結果：    ../
```


セキュリティトークン

Config/initializers/secret_token.rb

```
TestAPP::Application.config.secret_key_base = '0982380234...'
```

- ソースへの書き込みはNG（OSSでは必須）
  - CWE-798 SANS/CWE TOP 25 #7

```
TestAPP::Application.config.secret_key_base = ENV['SECRET_KEY_BASE']
```

必ず参照すべき資料

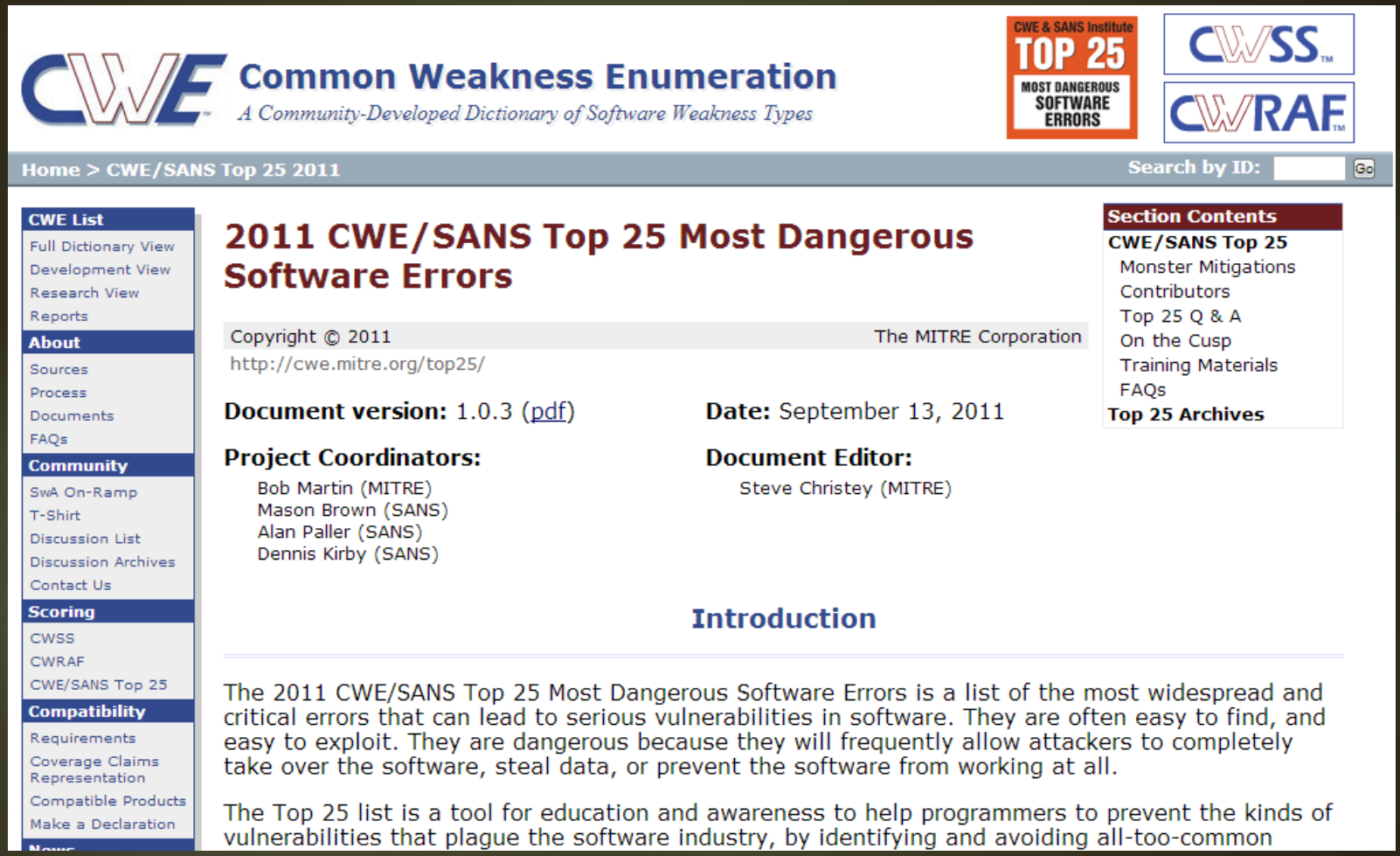
Railsセキュリティガイド

http://guides.rubyonrails.org/security.html

OWASP TOP 10

https://www.owasp.org/index.php/Top_10_2013-Top_10

SANS/CWE TOP 25



Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

CWE & SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS

CWSS

CWRAF

Home > CWE/SANS Top 25 2011

Search by ID: Go

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Contact Us
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration

2011 CWE/SANS Top 25 Most Dangerous Software Errors

Copyright © 2011 The MITRE Corporation
http://cwe.mitre.org/top25/

Document version: 1.0.3 (pdf)

Date: September 13, 2011

Project Coordinators:
Bob Martin (MITRE)
Mason Brown (SANS)
Alan Paller (SANS)
Dennis Kirby (SANS)

Document Editor:
Steve Christey (MITRE)

Section Contents
CWE/SANS Top 25
Monster Mitigations
Contributors
Top 25 Q & A
On the Cusp
Training Materials
FAQs
Top 25 Archives

Introduction

The 2011 CWE/SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical errors that can lead to serious vulnerabilities in software. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The Top 25 list is a tool for education and awareness to help programmers to prevent the kinds of vulnerabilities that plague the software industry, by identifying and avoiding all-too-common

http://cwe.mitre.org/top25/

Rails/Gemを信頼する？

Rails/Gemを信頼できるか？

ベストプラクティス「信頼できるフレームワーク・ライブラリ」利用

- 信頼するために検査することも要求
- SANS/CWE TOP 25　- 不正な外部コードの混入も
- Rails４　－　10000コミット以上、500以上のコントリビュータ

著名な開発者でもセキュリティスペシャリストとは限らない

- Rails3までのバリデーション機構はベストプラクティスとはいえない
- Mass Assingnment問題はPHPのregister_globals問題と同じ問題
- SANS/CWE TOP 25に記載されている脆弱性も考慮されていないことも
- セッション固定化の緩和策はとられていない（単独であれば小リスク）
- Ruby 2.1でscrub導入（不正文字エンコーディング対策）

誤解や混乱

- セキュリティ専門家と呼ばれる人達の間でも「正しい対策」について意見が分かれる
  - 例：文字エンコーディングバリデーション、セッション固定化対策

action_view¥helpers¥tag_helper.rb

- TagHelperに利用されるタグ属性処理

```
def tag_option(key, value, escape)
 value = value.join(" ") if value.is_a?(Array)
 value = ERB::Util.h(value) if escape
 %(#{key}="#{value}")
end
```

valueのみエスケープ

デフォルト有効

keyは通常シンボルなので「普通」は大丈夫？

Options

The `options` hash is used with attributes with no value like (`disabled` and `readonly`), which you can give a value of true in the `options` hash. You can use symbols or strings for the attribute names.

上級レベル開発者に必須

出力コントロール

- SANS/CWE TOP25 Monster Mitigation #2
  - **エスケープ → ヘルパー → バリデーション**

セキュリティ対策として順番

- セキュリティ対策を行うには出力先の「入力仕様」を正しく知る
- 「入力仕様」を知るには「**エスケープ処理**」を知る
- 「**ヘルパー**」の実装が正しいか、使い方が正しいか判断可能
- 「**バリデーション**」の実装が正しいか判断可能

フレームワーク・ライブラリとの付き合い方

妄信しない

- APIに制限があることは当たり前
- ベストプラクティスであるとは限らない

コードを確認する

正しいエスケープの方法を
知らないと判断できない

- APIを使う前に、さっと確認する

結果を確認する

正しいエスケープ結果を
知らないと判断できない

- 危ない入力でどんな結果になるのか確認する

ご清聴ありがとうございました。

WEBアプリのセキュリティのお問い合わせはINFO@ES-I.JPへ