Webセキュリティの基本のキ

EG セキュアソリューションズ株式会社
取締役CTO 徳丸 浩

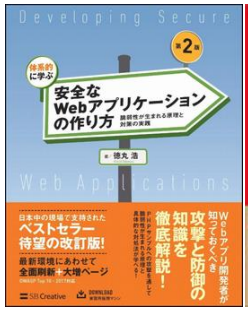
徳丸浩のウェブセキュリティ講座

アジェンダ

- セッション管理
- 認可制御不備
- CSRF
- クロスサイトスクリプティング
- SQLインジェクション

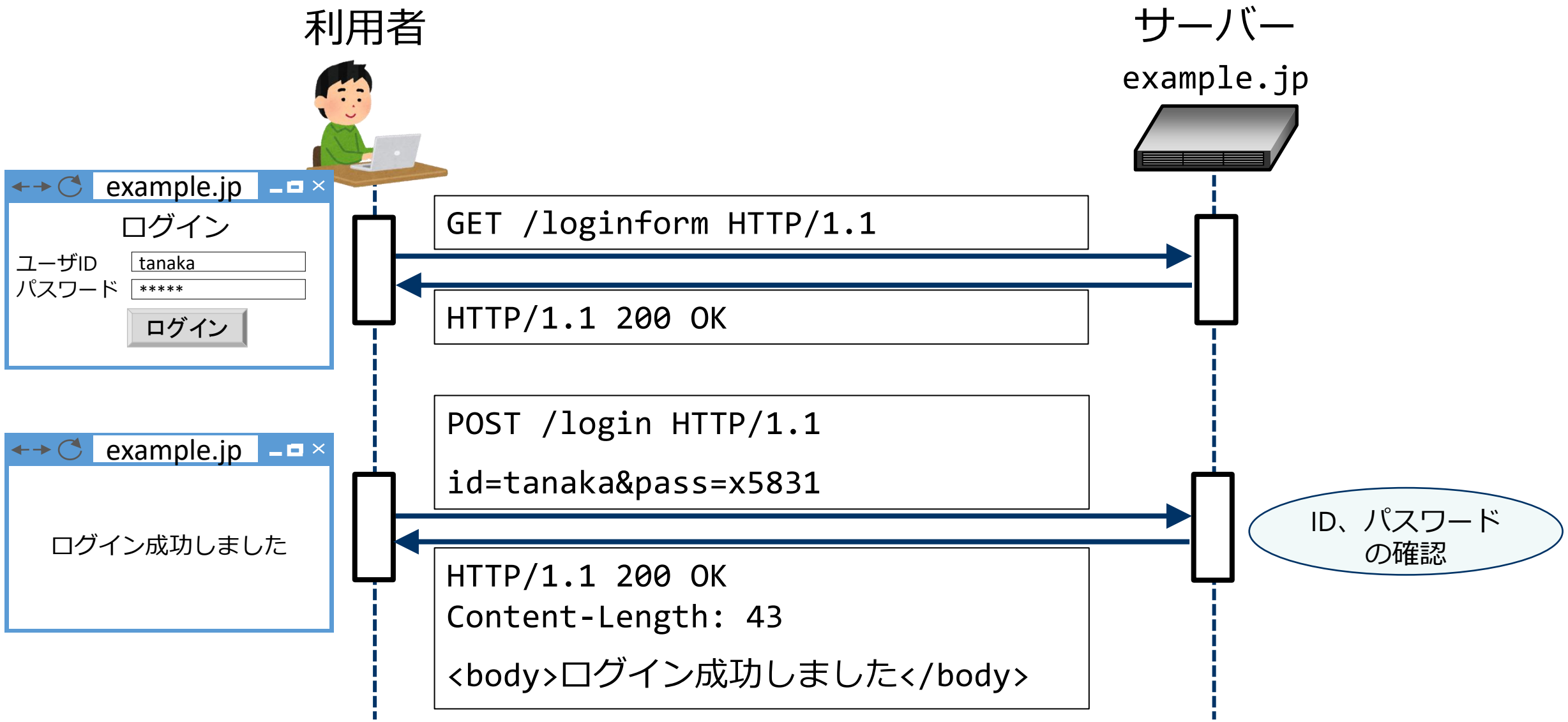
徳丸浩の自己紹介

- 経歴
  - 1985年 京セラ株式会社入社
  - 1995年 京セラコミュニケーションシステム株式会社(KCCS)に出向・転籍
  - 2008年 KCCS退職、HASHコンサルティング株式会社(現社名:EGセキュアソリューションズ株式会社)設立
- 経験したこと
  - 京セラ入社当時はCAD、計算幾何学、数値シミュレーションなどを担当
  - その後、企業向けパッケージソフトの企画・開発・事業化を担当
  - 1999年から、携帯電話向けインフラ、プラットフォームの企画・開発を担当
    Webアプリケーションのセキュリティ問題に直面、研究、社内展開、寄稿などを開始
  - 2004年にKCCS社内ベンチャーとしてWebアプリケーションセキュリティ事業を立ち上げ
- 現在
  - EGセキュアソリューションズ株式会社取締役CTO　　https://www.eg-secure.co.jp/
  - 独立行政法人情報処理推進機構　非常勤研究員　　https://www.ipa.go.jp/security/
  - 著書「体系的に学ぶ 安全なWebアプリケーションの作り方（第2版）」(2018年6月)
  - YouTubeチャンネル「徳丸浩のウェブセキュリティ講座」
    https://j.mp/web-sec-study
  - 技術士（情報工学部門）



セッション管理

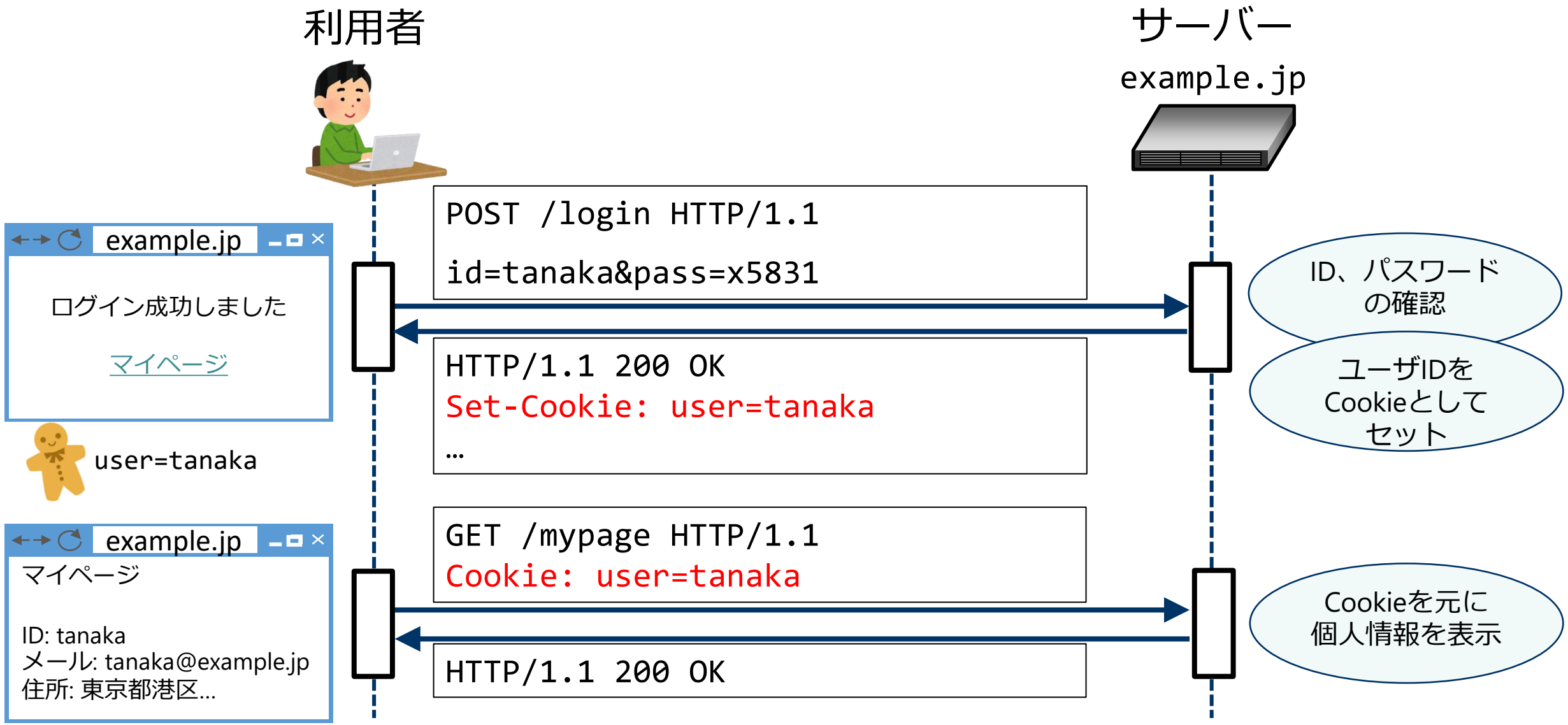
ログイン処理の様子



ログイン結果の状態をどう保存するか

- ウェブが使用するプロトコルHTTPは「状態」を保存しない
- ログイン結果を以降のページにも保存したい
- ブラウザには「状態」を保存する機能が2種類ある
  - Cookie: 伝統的な方法
    - Set-Cookie レスポンスヘッダ：Cookieの保存をブラウザに命令
      `Set-Cookie: name=yamada`
    - Cookie リクエストヘッダ：保存したCookieを自動的にサーバーに送信
      `Cookie: name=yamada`
  - ウェブストレージ（ localStorage等）：JavaScriptから利用できる
    - `setItem`メソッド：`localStorage`への保存
      `localStorage.setItem('name', 'yamada')`
    - `getItem`メソッド：`localStorage`からの値取り出し
      `const name = localStorage.getItem('name')  // name`には`'yamada'`が入る

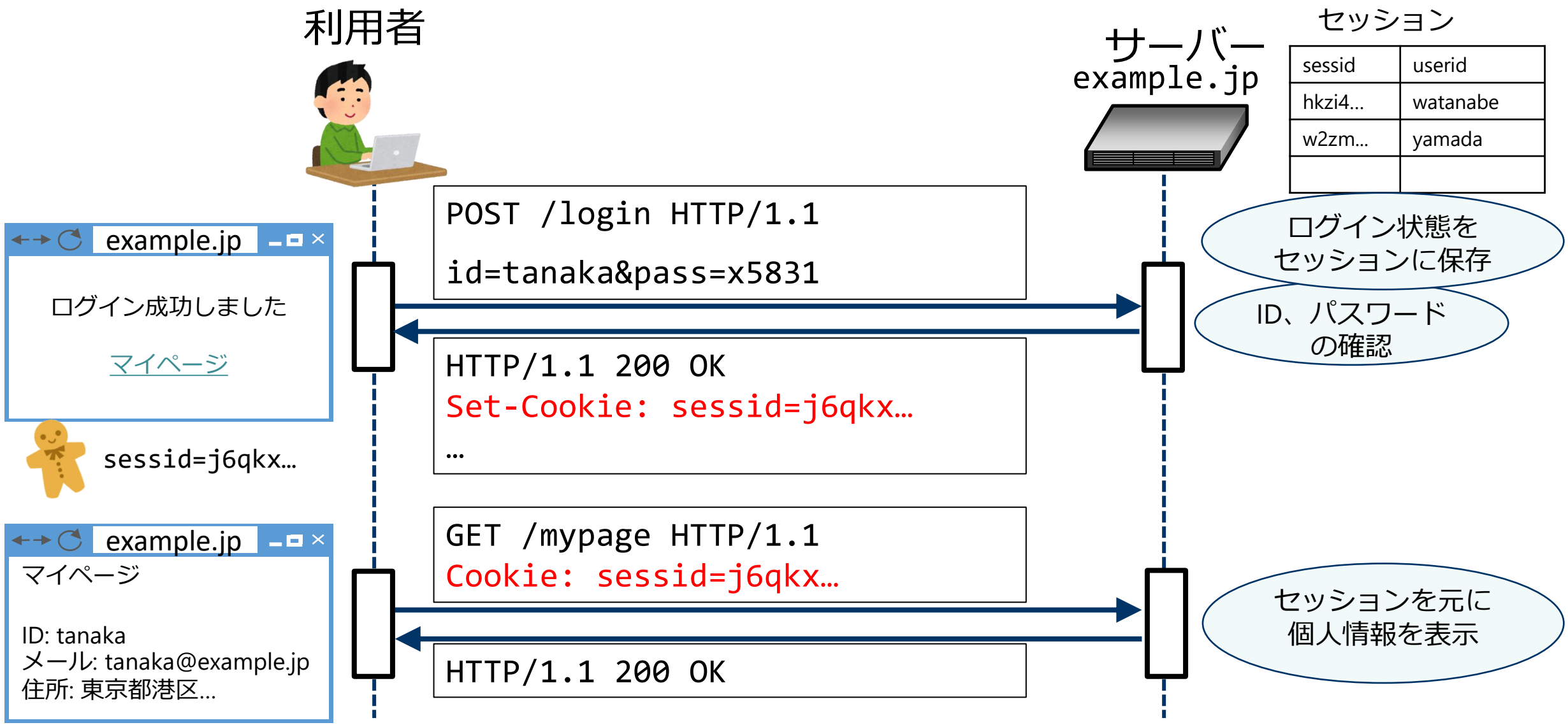
ログイン後Cookieにユーザidを保存（脆弱な処理）



実は、これだと問題が…

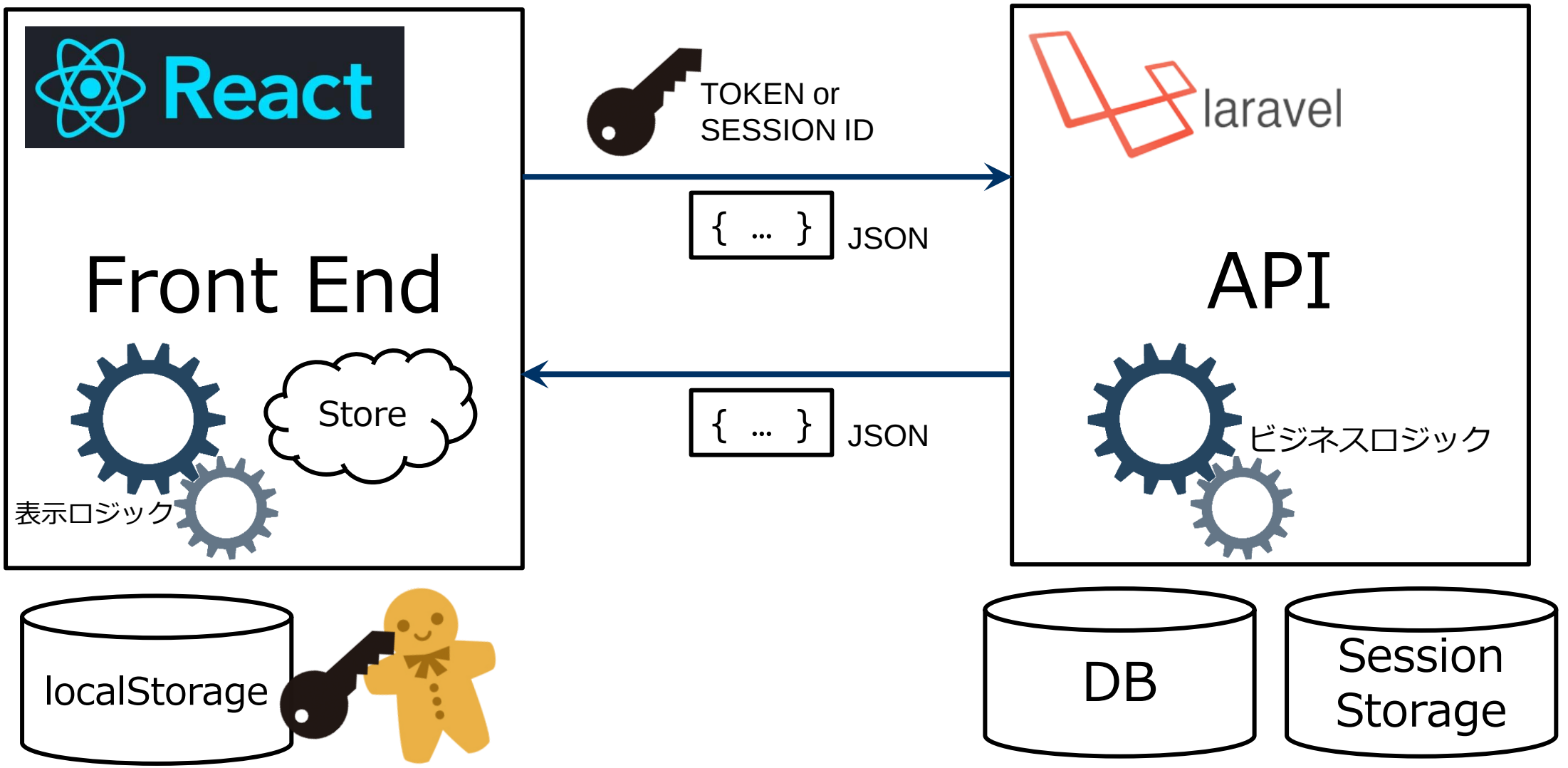
機密情報のCookieへの直接保存(CWE-315)

セッション管理機能を使いましょう

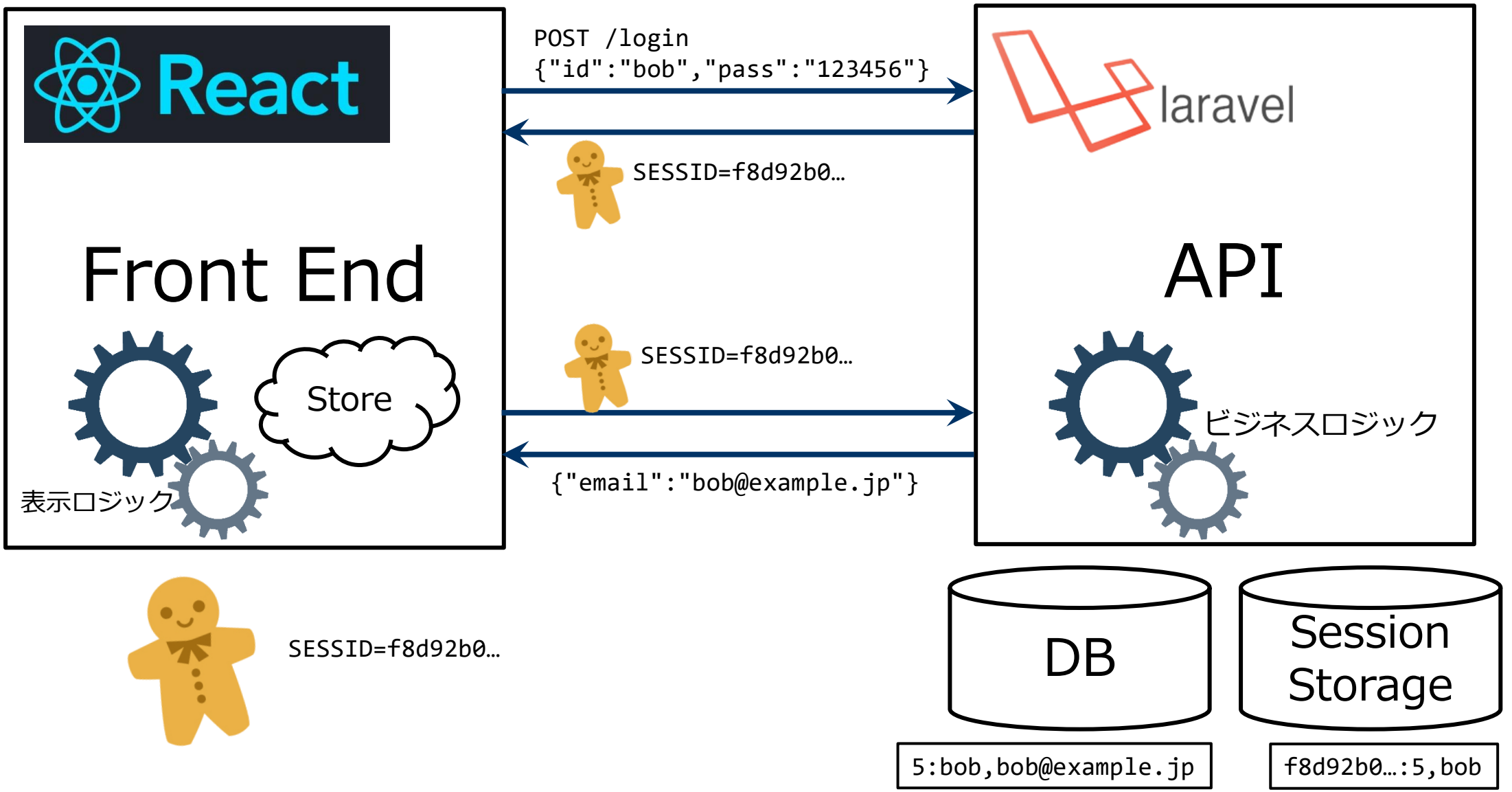


SPAと攻撃経路

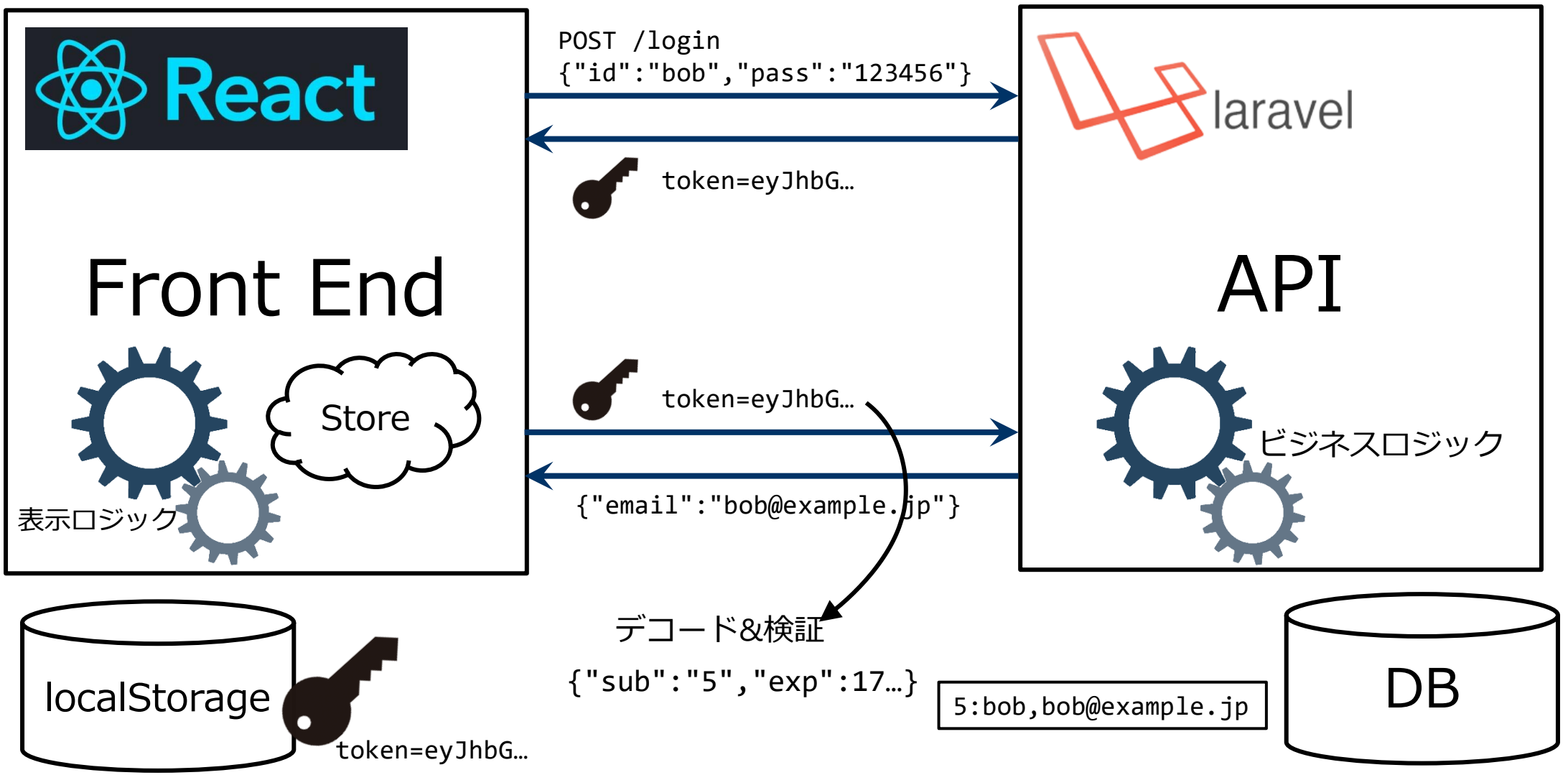
SPAの構造



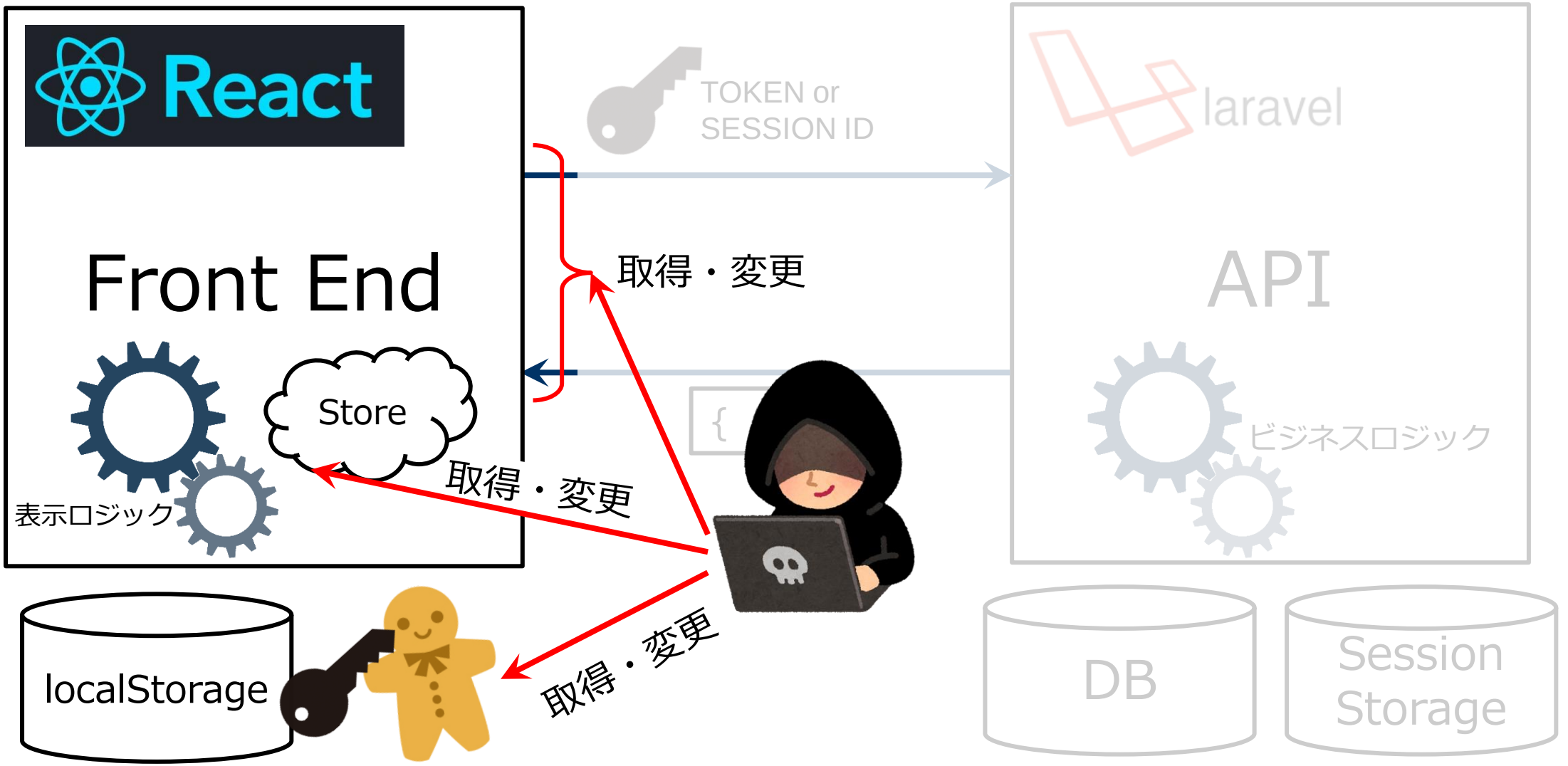
SPAのセッション管理方式 1 セッションID方式



SPAのセッション管理方式 2 ステートレストークン(JWT等)

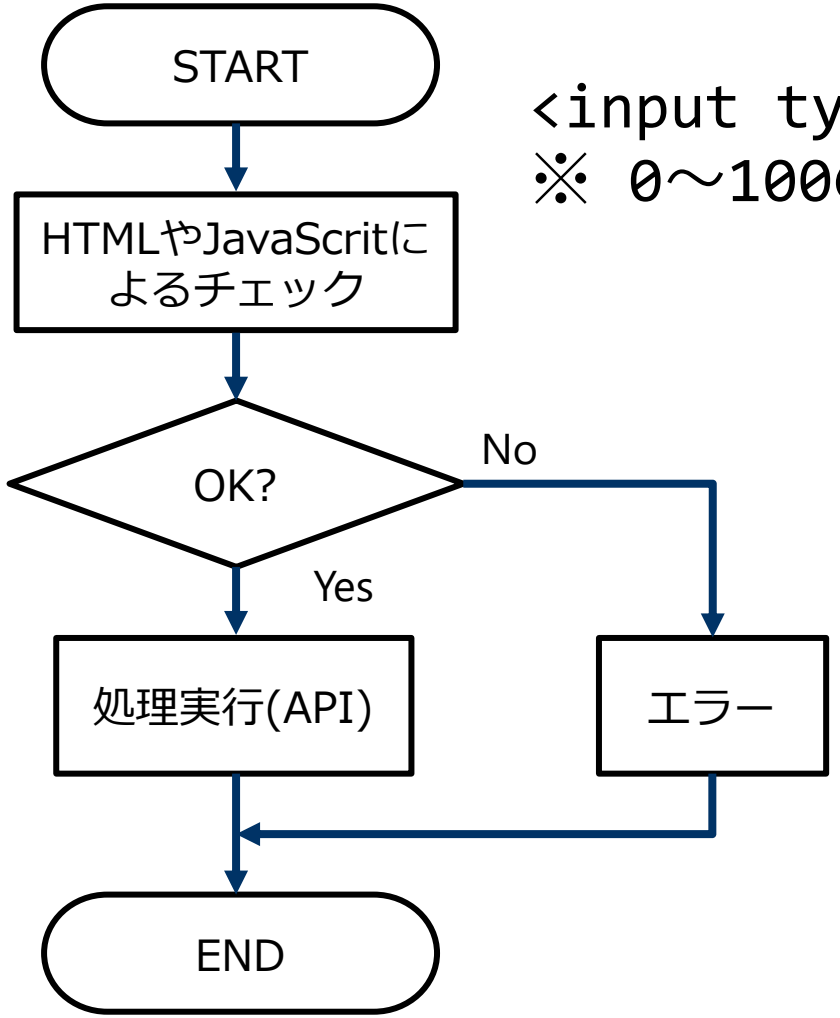


クライアント側のロジックやデータは取得・変更可能



JavaScriptによる検証はバイパスできる

悪用例1: クライアント側バリデーションの回避

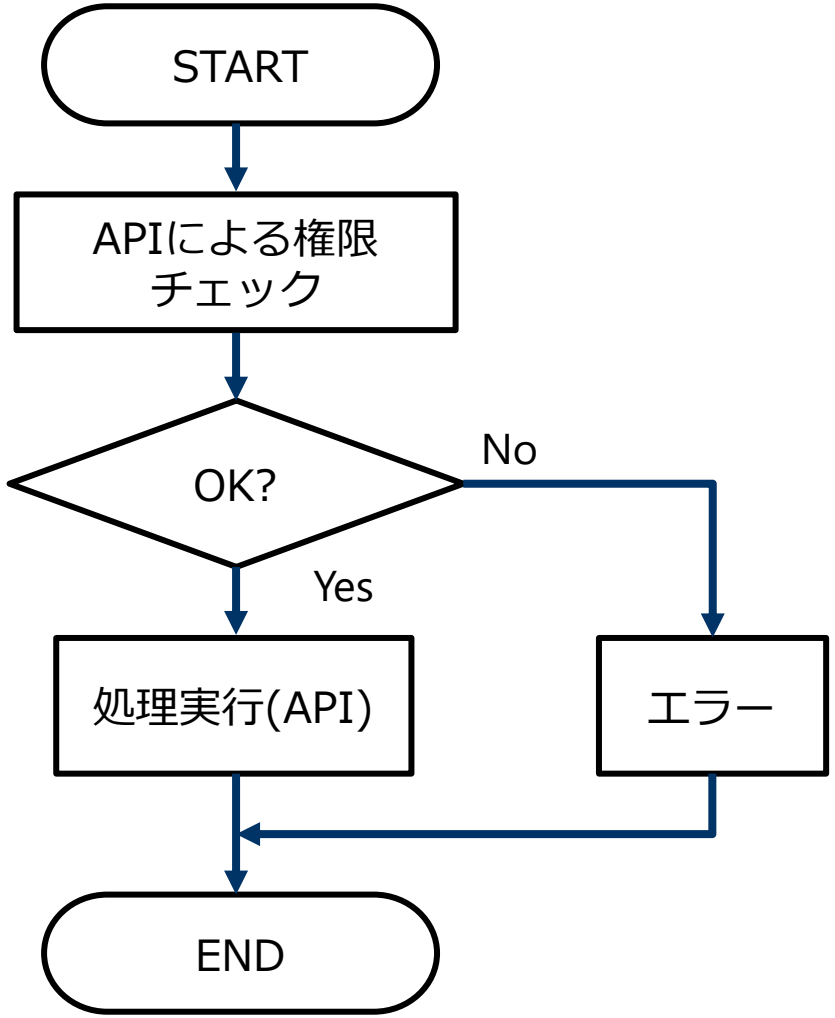



```
<input type="number" min="0" max="100">
```

※ 0～100の数値のみ許容

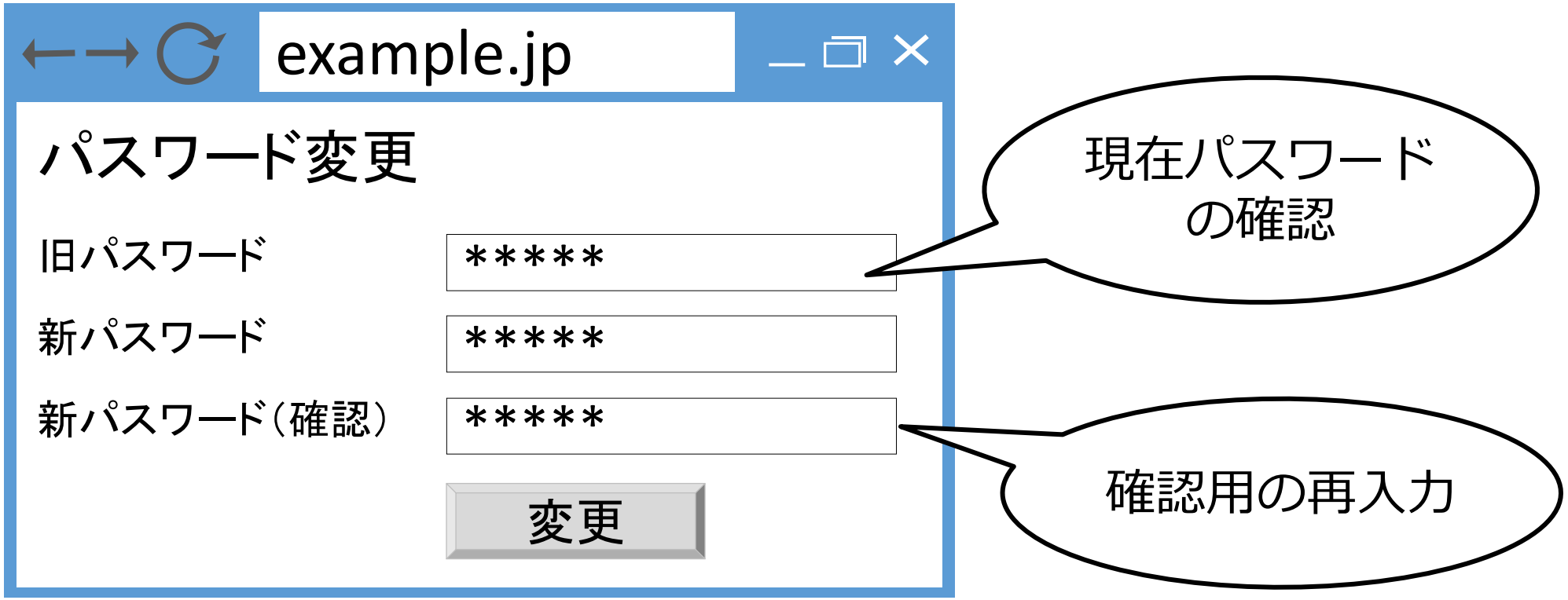
HTMLやJavaScriptによるチェック（バリデーション等）は攻撃者は回避可能

悪用例2: クライアントによるチェック処理の回避

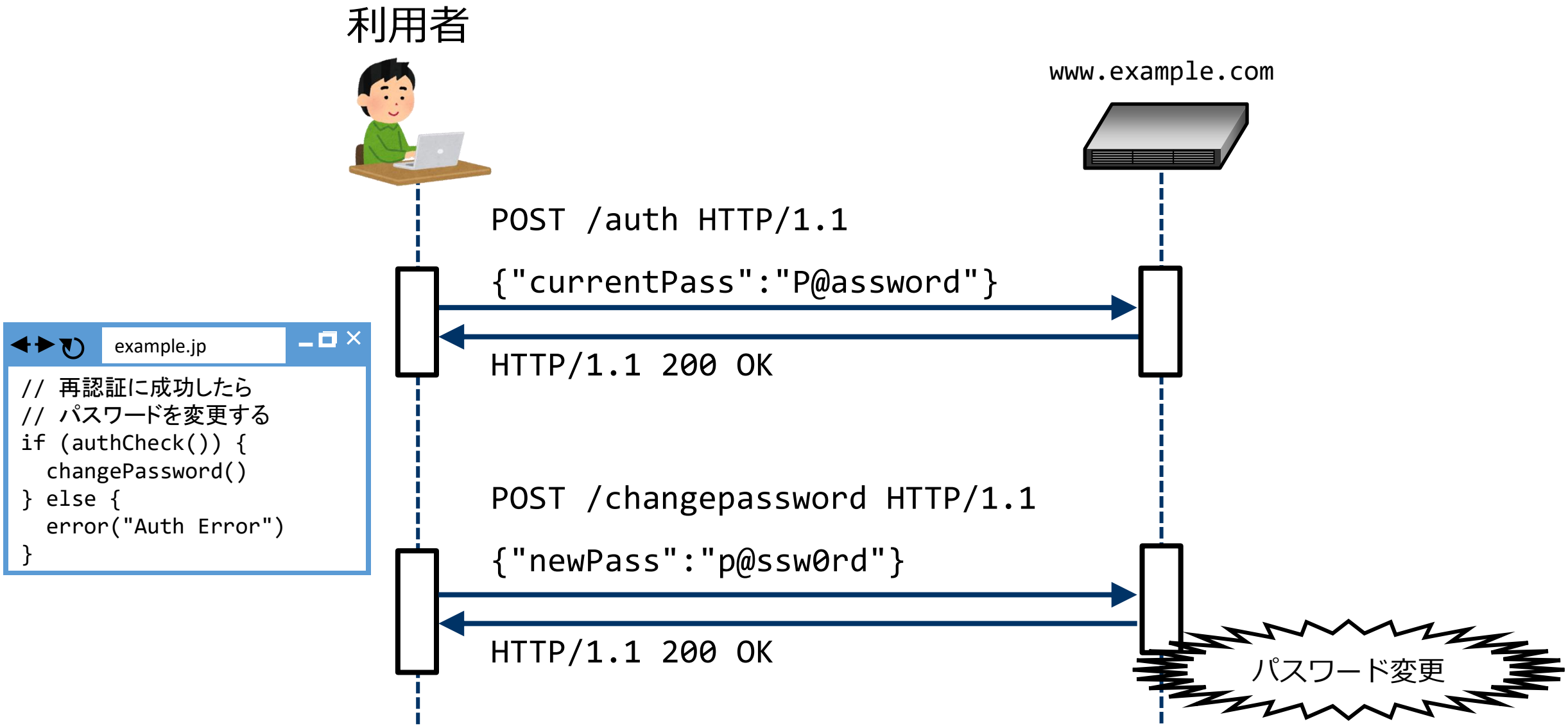


APIによる処理でも、チェックと実行（更新等）が別だと、チェック処理は回避可能

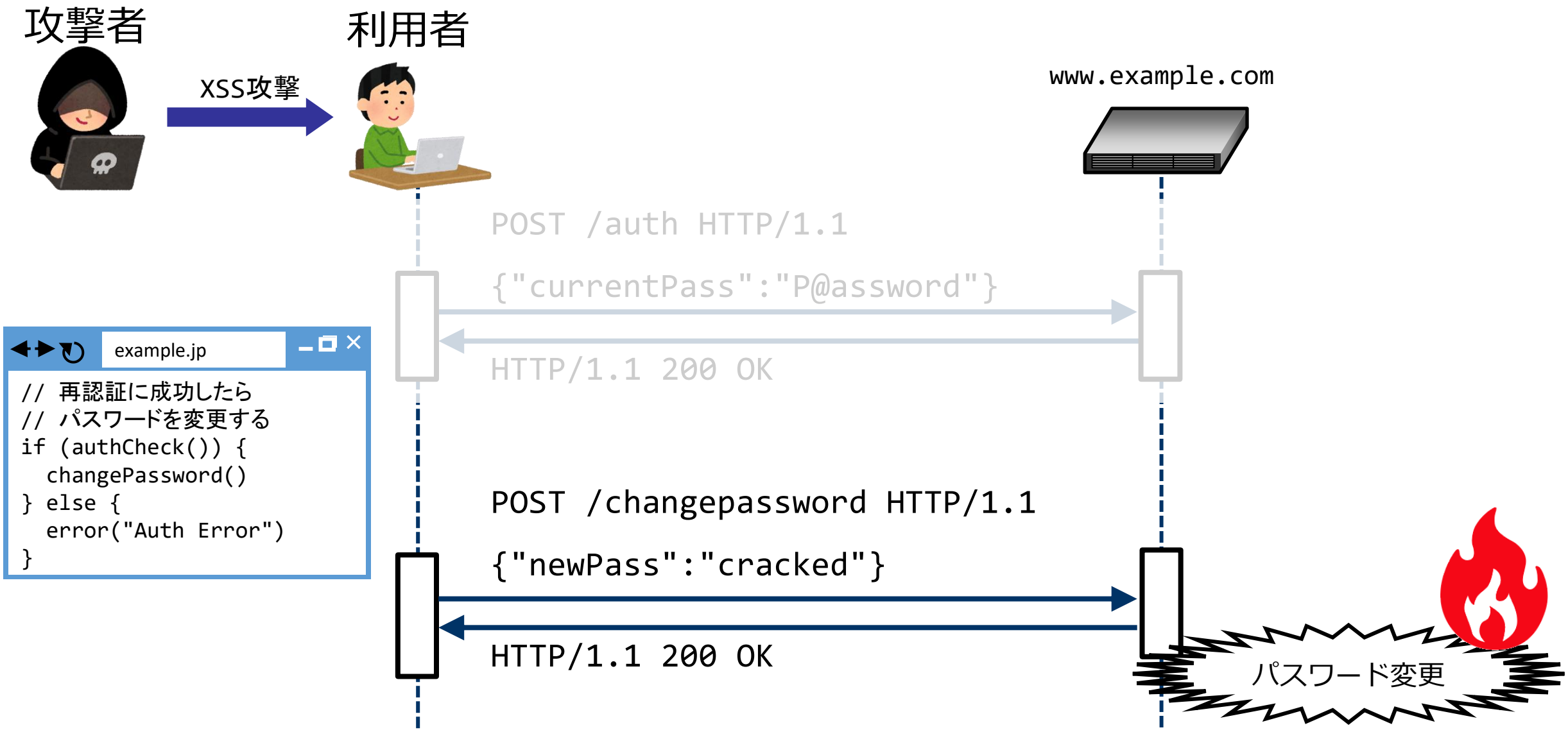
パスワード変更機能の場合



パスワード変更機能の脆弱な再認証の実装



パスワード変更機能の脆弱な再認証の実装



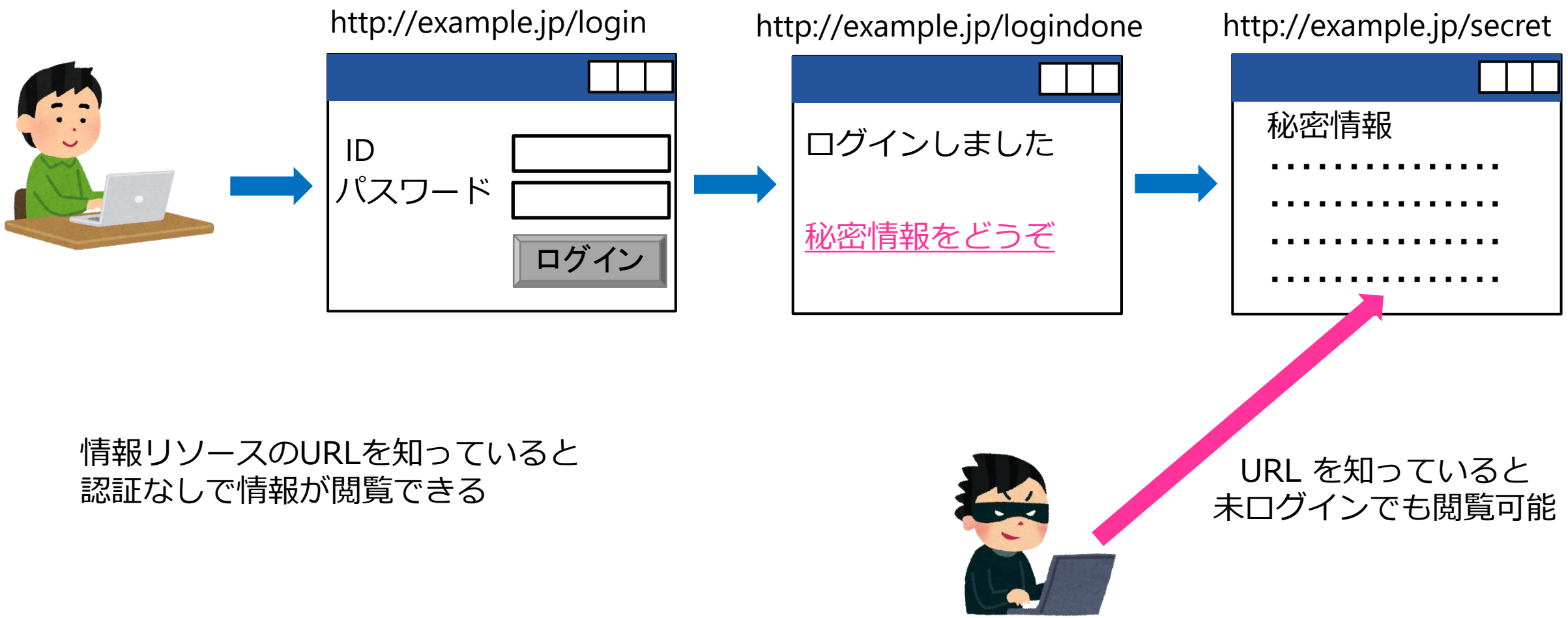
認可制御不備

認可制御とは

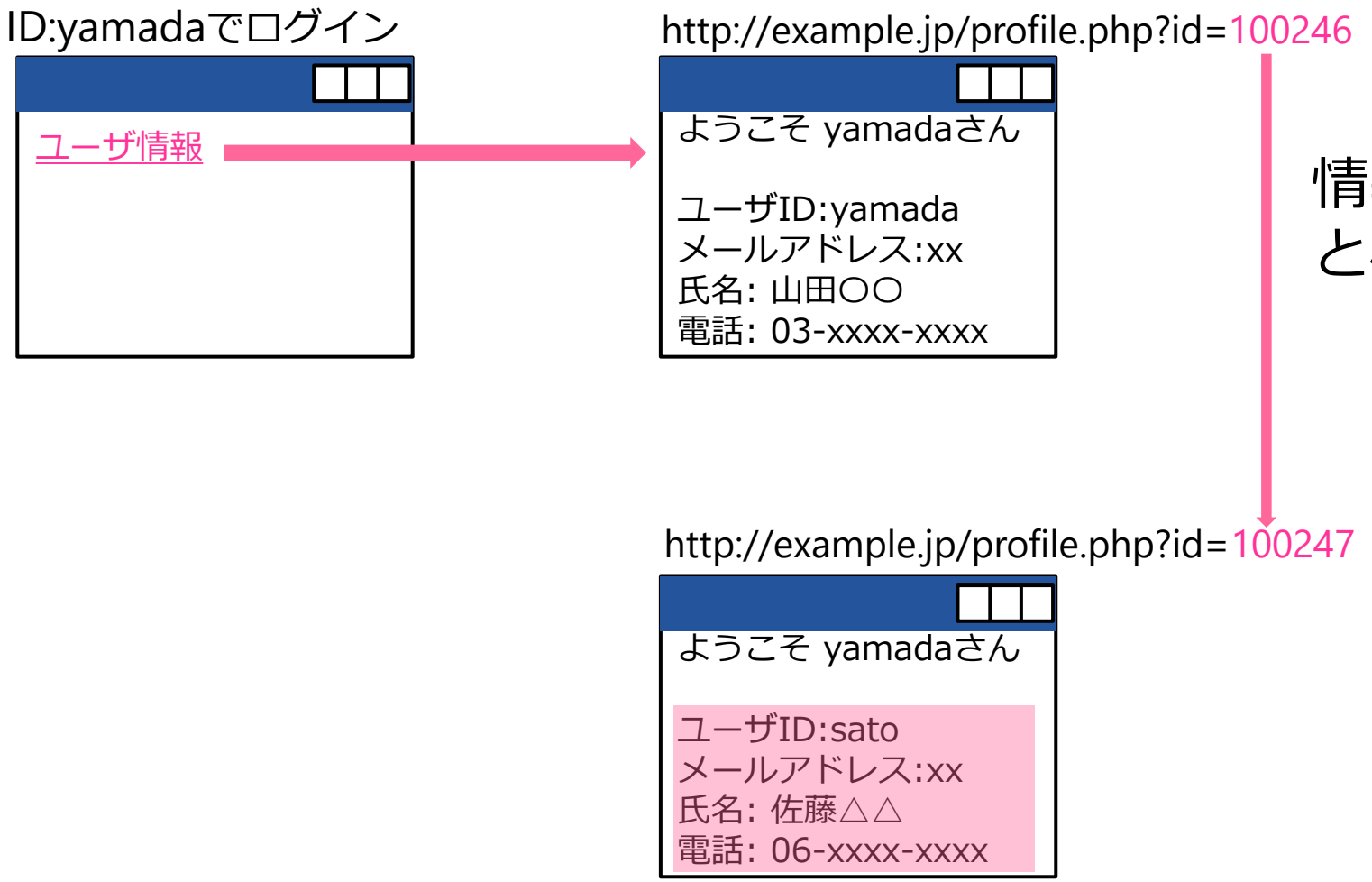
- 認証(Authentication): ユーザーが誰かを確認すること
- 認可(Authorization): 認証されたユーザーに対して許可（権限）を与えること
- アクセス制御(Access Control): 認証しているユーザーのみにアクセスを許可すること

注) アクセス制御は、IPアドレス等による制御を含む場合もある

認可制御不備の典型的パターン(1)



認可制御不備の典型的パターン(2)

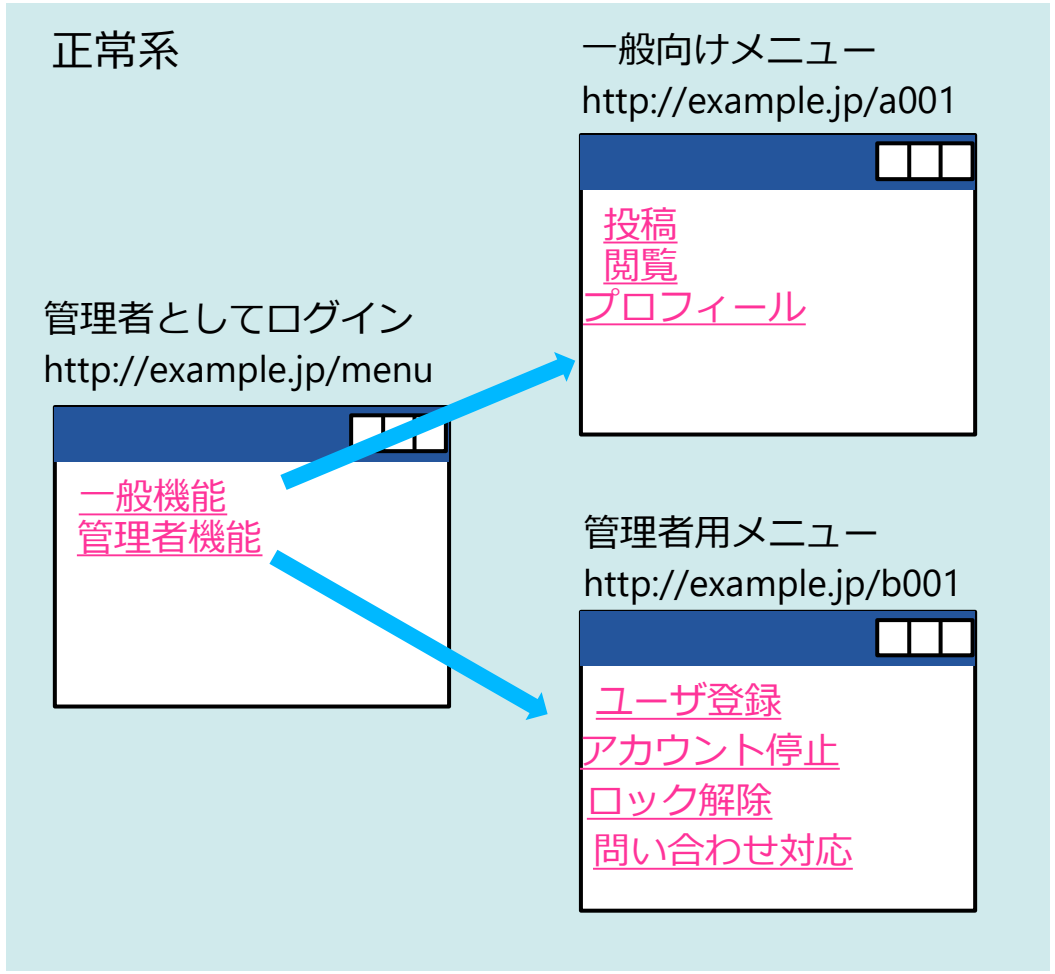


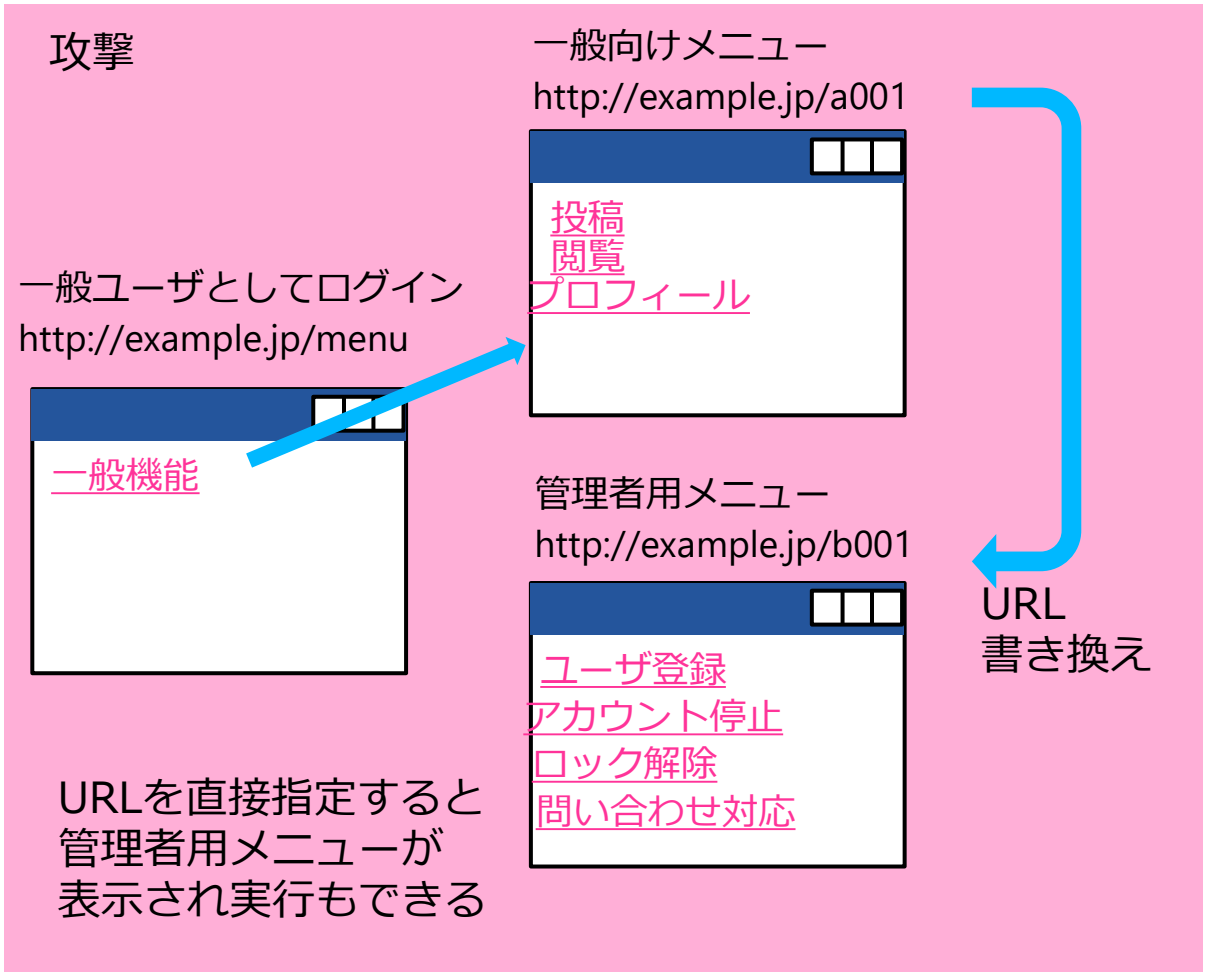
情報リソースのIDを変更すると権限外の情報が参照できる

内部ID	外部ID
100246	yamada
100247	sato

※アプリケーション内部では表示用の外部IDではなく、内部ID(一連番号)で管理されている

認可制御不備の典型的パターン(3)



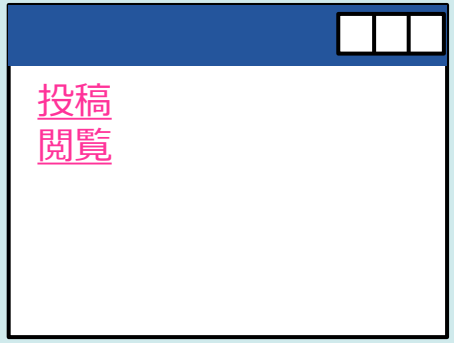


メニューの表示・非表示のみで制御している

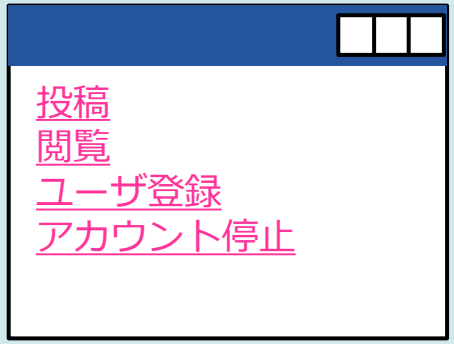
認可制御不備の典型的パターン(4)

正常系

一般ユーザとしてログイン
Cookie: USER=normal

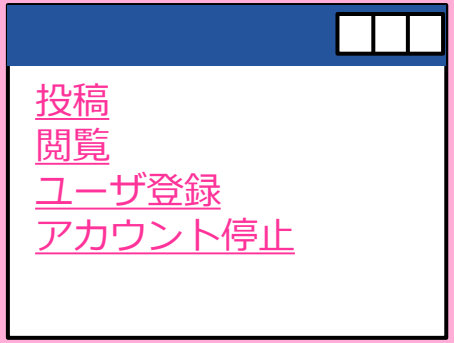


管理者としてログイン
Cookie: USER=admin



攻撃

一般ユーザとしてログイン
Cookie: USER=admin



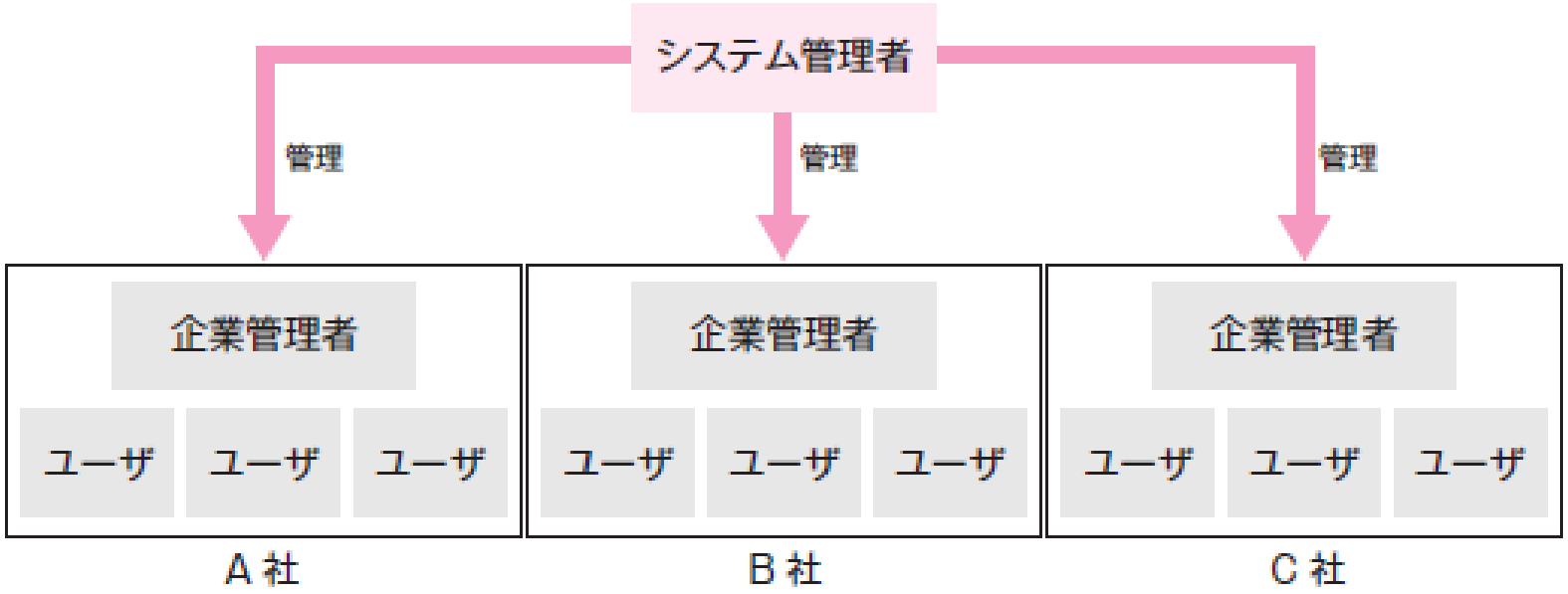
Cookieに保持した権限情報を変更しただけで管理者メニューが表示され、実行もできる

hiddenパラメータやクッキーに権限情報を保持している

認可制御の正しい実装

- 認可制御の基本は、以下の直前に権限を確認すること
  - 秘密情報の表示
  - 権限の必要な機能実行
  - その他、権限を必要とする操作
- 権限を確認する基準は、セッション変数に保存したログインID
- admin等の特権専用ユーザーは作らず、個人IDに権限を割り当てる仕様が好ましい
- 複雑な認可制御が必要な場合は、役割を抽象化したロールを定義する
- 時間経過に伴い、権限やロールが変化する場合もあるので、権限やロールはセッション変数に保存せず、都度確認する
- ロールと権限を「権限マトリックス」としてドキュメント化する

ロールと権限マトリックスの例

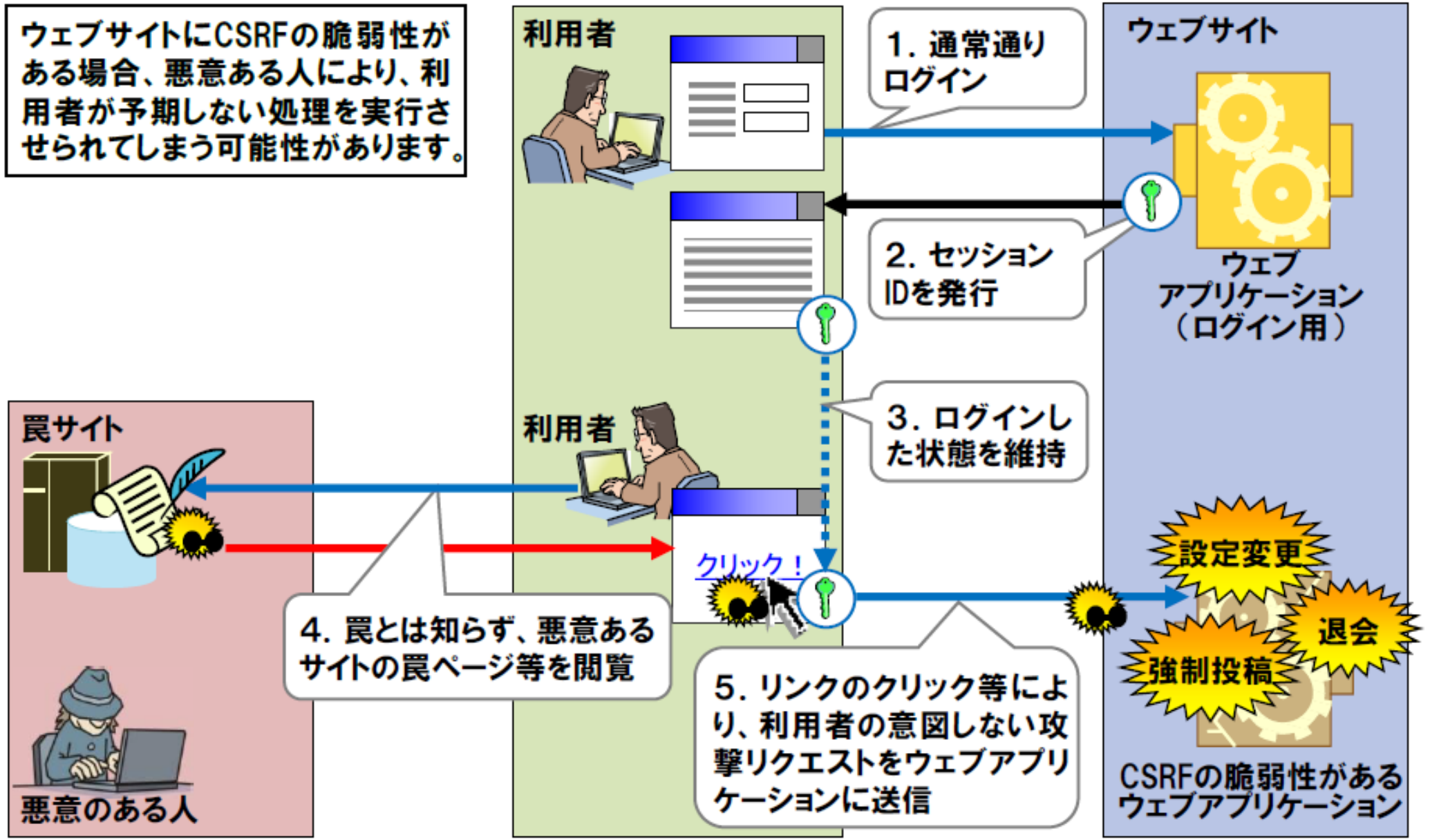


権限

ロール	システム管理者	企業管理者	一般ユーザ
企業の追加	○	×	×
企業管理者の追加・削除	○	×	×
企業内ユーザの追加・削除	○	○	×
自分自身のパスワード変更	○	○	○
他人のパスワード変更	○	企業内に限り○	×

1.クロスサイト・リクエストフォージェリ(CSRF)

クロスサイト・リクエストフォージェリ(CSRF)とは



https://www.ipa.go.jp/security/vuln/websecurity.html より引用

クロスサイト・リクエストフォージェリの原理

正常系のHTTPリクエスト

```
POST /45/45-003.php HTTP/1.1
Referer: http://example.jp/45/45-002.php
Content-Type: application/x-www-form-urlencoded
Host: example.jp
Cookie: PHPSESSID=isdv0mecsobejf2oalnuf0r1l2
Content-Length: 9

pwd=pass1
```

CSRF攻撃時のHTTPリクエスト

```
POST /45/45-003.php HTTP/1.1
Referer: http://trap.example.com/45/45-900.html
Content-Type: application/x-www-form-urlencoded
Host: example.jp
Cookie: PHPSESSID=isdv0mecsobejf2oalnuf0r1l2
Content-Length: 9

pwd=pass1
```

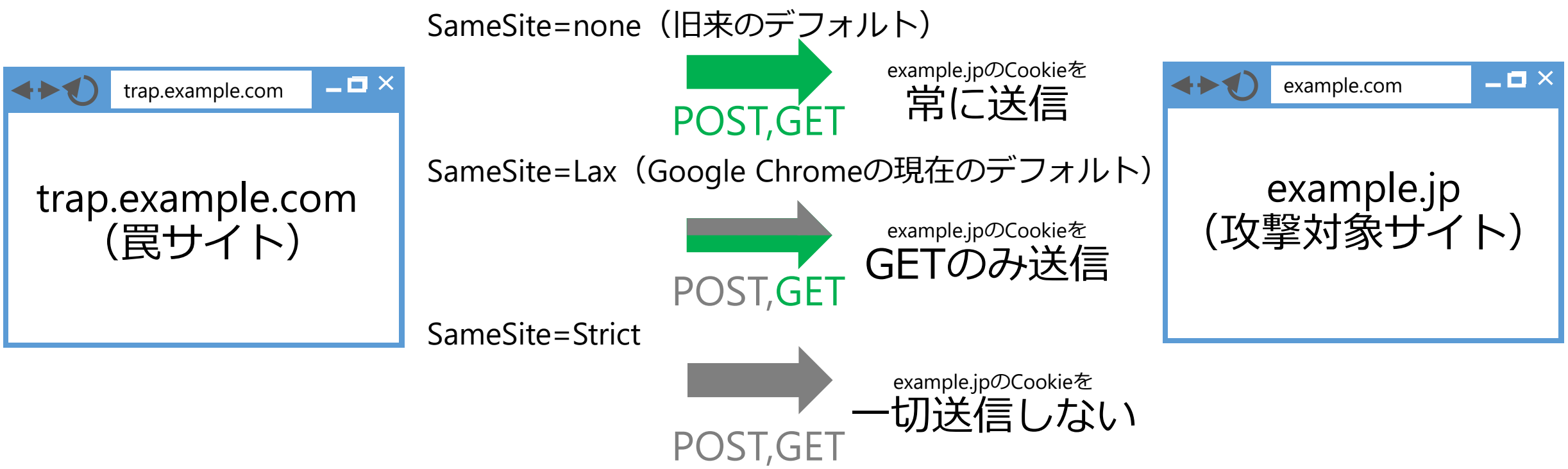
クロスサイト・リクエストフォージェリの影響と対策

- 影響
  - 脆弱性のある機能を第三者に強制される
  - パスワード変更
  - 勝手な投稿、変更、削除
  - 物品購入
- 対策
  - 方法1 トークン(Token)による方法（推奨）
    ※ アプリケーションフレームワークのCSRF対策機能を素直に使う
  - 方法2 パスワード確認（再認証）
  - 保険的対策：重要な処理実行後に登録済みメールアドレスに通知を送付する

補足：CookieのSameSite属性について

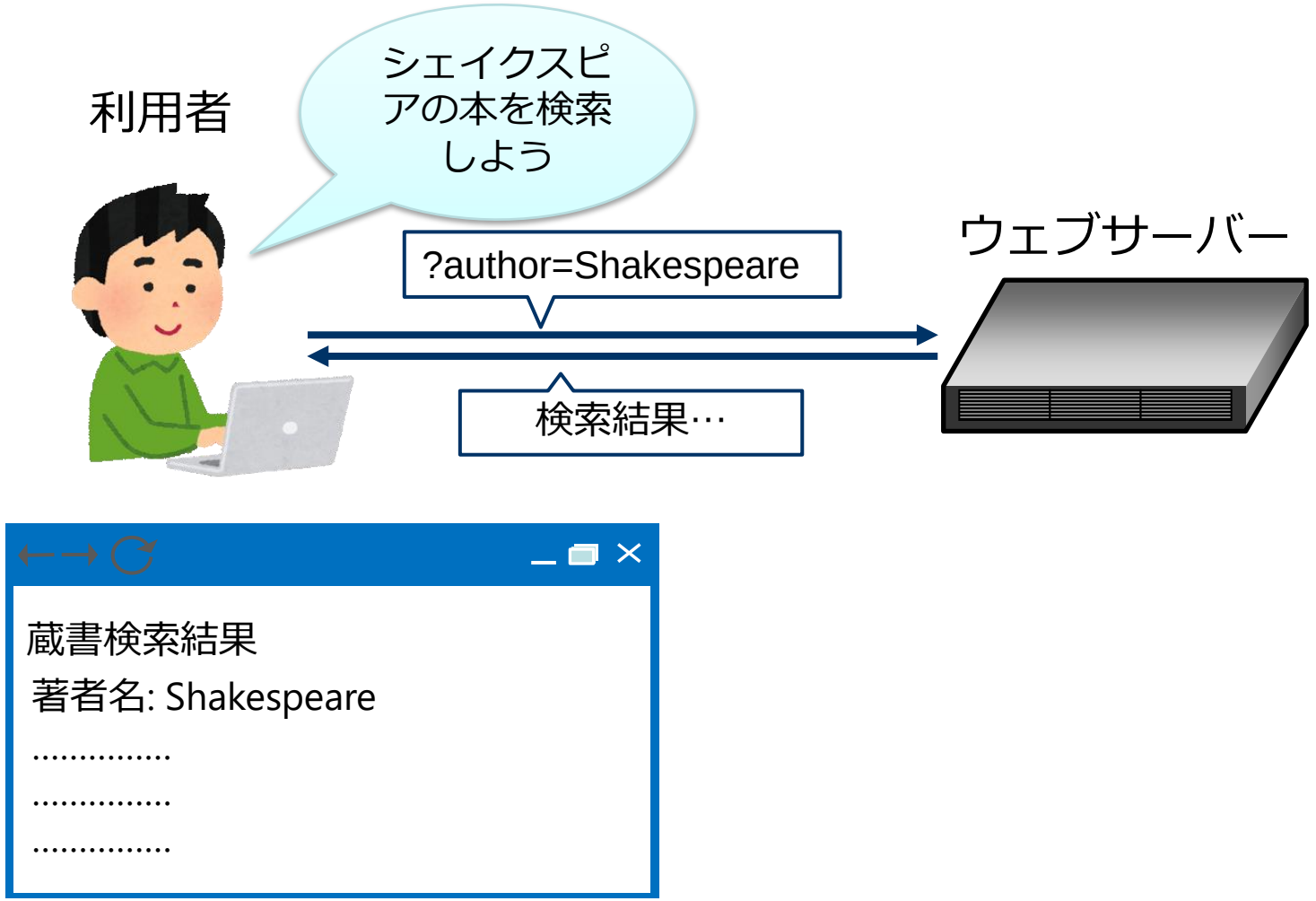
- SameSite属性は、異なるサイトから遷移した場合に、クッキーを送信するかどうかを制御するもの

元々Cookieがセットされていた場合に、Cookieが送信されるか



2.クロスサイトスクリプティング(XSS)

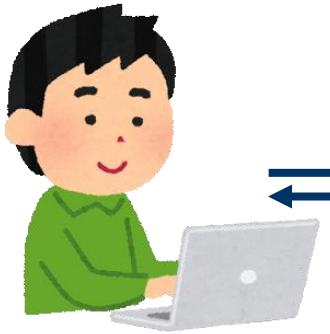
クロスサイトスクリプティングのイメージ（正常系）



クロスサイトスクリプティングのイメージ（攻撃）

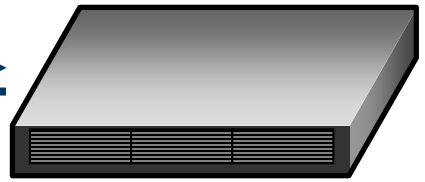
図書館からのお知らせ
○○様、至急ご確認ください
https://xxxxxxx/?author=<sci

利用者



?author=<script>…

ウェブサーバー



検索結果…
<script> …


```
<h1>蔵書検索結果</h1>
<p>著者名<script> /* 攻撃コード */ </script>
```

クロスサイトスクリプティングの影響

- 攻撃を受けた人のパソコンが遠隔操作される
  - なりすまし投稿
  - なりすましの買い物
  - 情報漏えい
- 影響は、脆弱性のあるサイト全体に及びます
  - 「重要でない」ページに影響があっても、個人情報漏洩なども起こりえる
- 他のサイトには直接影響はない
- 攻撃を受けた人（サイトを閲覧した人）のみに影響は限られる

対策

XSSの対策（根本的解決策）

- 文脈に応じてHTMLエスケープ

```
< → &lt;        > → &gt;
& → &amp;       " → &quot;
' → &#39; または &apos;
```

- 属性値はダブルクオートで囲む
- URLを取る属性値はスキームのチェックを行う
  - http: または https: または /
- レスポンスヘッダで文字エンコーディングを指定
  - Content-Type: text/html; charset=UTF-8
- JavaScriptの動的生成を避ける
  - HTML上に値を書いてJavaScriptから参照
  - JavaScriptのエスケープは人類には難しすぎる

対策後のプログラム（PHP）

```
<?php
  session_start();
  // ログインチェック（略）
?>
<body>
検索キーワード:<?php echo htmlspecialchars($_GET['keyword']); ?><BR>
以下略
</body>
```

クエリ文字列として <script>alert(document.cookie)</script> が入力された場合、表示は以下となる

検索キーワード:<script>alert(document.cookie)</script>

XSSの対策（保険的対策）

- CookieにHttpOnly属性を付与する
  - CookieにHttpOnly属性を付与すると、JavaScriptからCookieにアクセス禁止に
  - 他の方法で攻撃できるので、効果は限定的
- X-XSS-Protection 以下を設定する
  - `X-XSS-Protection: 1; mode=block`

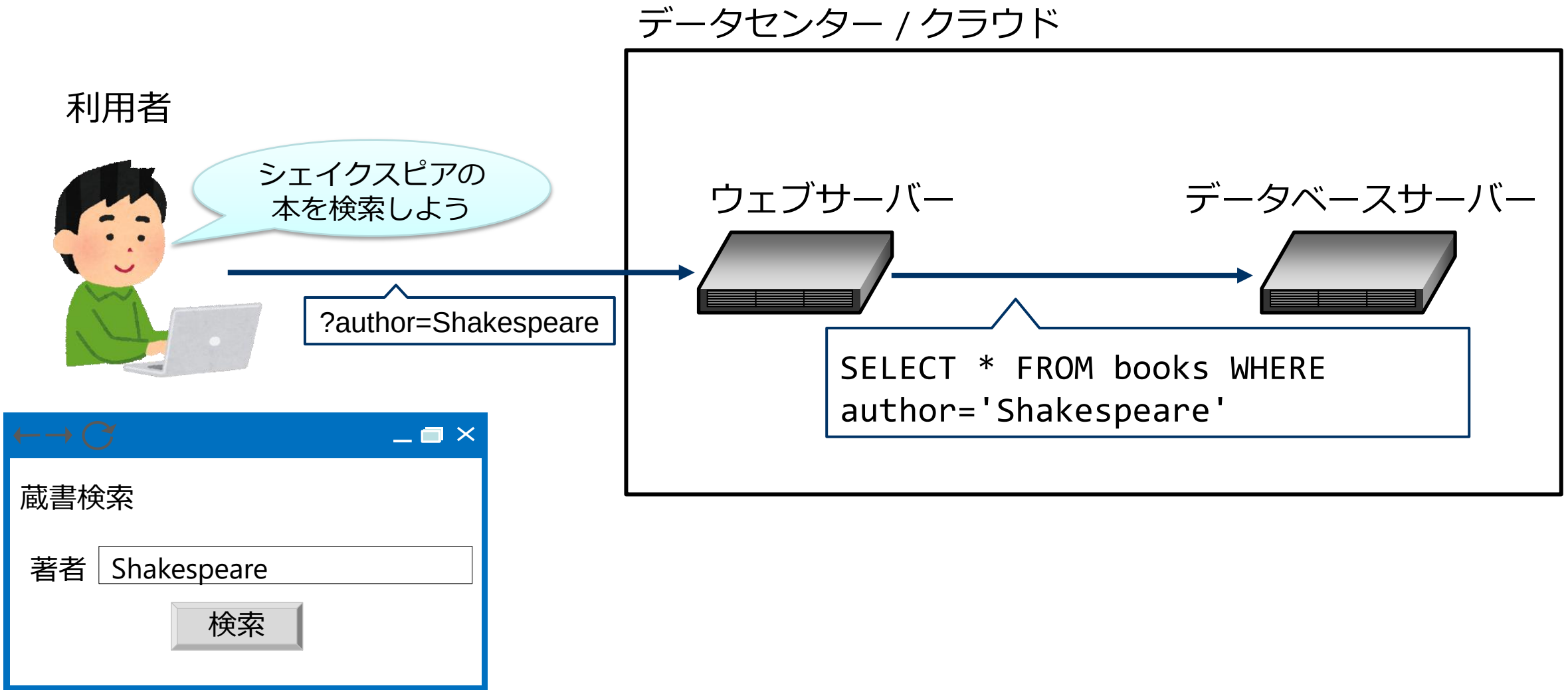
  ※ Google Chrome、Firefox、Safariでは無効化されている
- TRACEメソッドの無効化
  - XSSの応用としてCross Site Tracingがあり、TRACEメソッドの無効化により対策する（ただし、ブラウザ側で対策されているので実害はない）
  - Apacheでの設定: `TraceEnable Off`
  - Nginxでは元々無効化されているので対策不要

まとめ

- ブラウザには、サンドボックス、同一生成元ポリシー(SOP)などの保護機能がある
- SOPの保護機能により、正規サイトと「怪しいサイト」はアクセスが遮断されている
- XSSは、SOPの保護機能の元で、正規サイトの情報に、「怪しいサイト」からアクセスできる（これはアプリケーションの脆弱性）
- XSSがあると、ワナサイトを閲覧した利用者の個人情報が漏洩
  – あるいは「なりすまし犯行予告」の書き込みをさせることも…
- XSS対策の基本は、「終端」となる記号の文脈に応じたエスケープ
- でも、色々ややこしいので参考文献で勉強を

3.SQLインジェクション

SQLインジェクションのイメージ（正常系）



SQLインジェクションのイメージ（攻撃）

データセンター / クラウド

利用者

ウェブサーバー

データベースサーバー

?author=';DELETE+FROM+
books+--+

```
SELECT * FROM books WHERE name='';
DELETE FROM books -- '
```

蔵書検索

著者 ';DELETE FROM books--

検索

DELETE文により
booksテーブルの
中身が削除される

SQLインジェクションの影響

- データベース内のすべての情報が漏洩する
  - アプリケーションがSQL呼び出しに使うデータベースユーザの権限による
- データの改変、削除
- データベースサーバー内のファイルの読出し、書込み（設定による）
- OSコマンドの実行（設定による）
- 認証機能の回避

SQLインジェクションの対策

- 根本的解決策
  - SQL文を動的に組み立てない
  - プレースホルダによりSQL文を組み立てる
  - フレームワークのO/Rマッパーなどを正しく使う

- 保険的対策
  - 入力値の妥当性検証
  - 詳細なエラーメッセージの抑止
  - データベースのアクセスユーザに必要最低限の権限を割り当てる

プレースホルダによる対策例（PHP）

```php
<?php
  $author = $_GET['author'];  // クエリ文字列から著者名を取得
  $db = … // データベース接続
  // 静的プレースホルダを使う設定
  $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
  // ? はプレースホルダ
  $sql = "SELECT * FROM books WHERE author = ?";
  $dbh = $db->prepare($sql);  // SQLを準備する
  $dbh->bindValue(1, $author, PDO::PARAM_STR); // バインド
  $ps = $db->execute();  // クエリ実行
```

ご清聴ありがとうございました