「Javaアプリケーション脆弱性事例調査資料」について

- この資料は、Javaプログラマである皆様に、脆弱性を身近な問題として感じてもらい、セキュアコーディングの重要性を認識していただくことを目指して作成しています。

- 「Javaセキュアコーディングスタンダード CERT/Oracle版」と合わせて、セキュアコーディングに関する理解を深めるためにご利用ください。

JPCERTコーディネーションセンター
セキュアコーディングプロジェクト
secure-coding@jpcert.or.jp

Spacewalkにおけるクロスサイトリクエストフォージェリ(CSRF)の脆弱性

CVE-2009-4139

JVNDB-2011-003800

Spacewalkとは

- SpacewalkはWebベースのLinuxシステムの統合管理ツール
- Red Hat社がリリースしている「Red Hat Network Satellite」のオープンソース版

Spacewalkとは

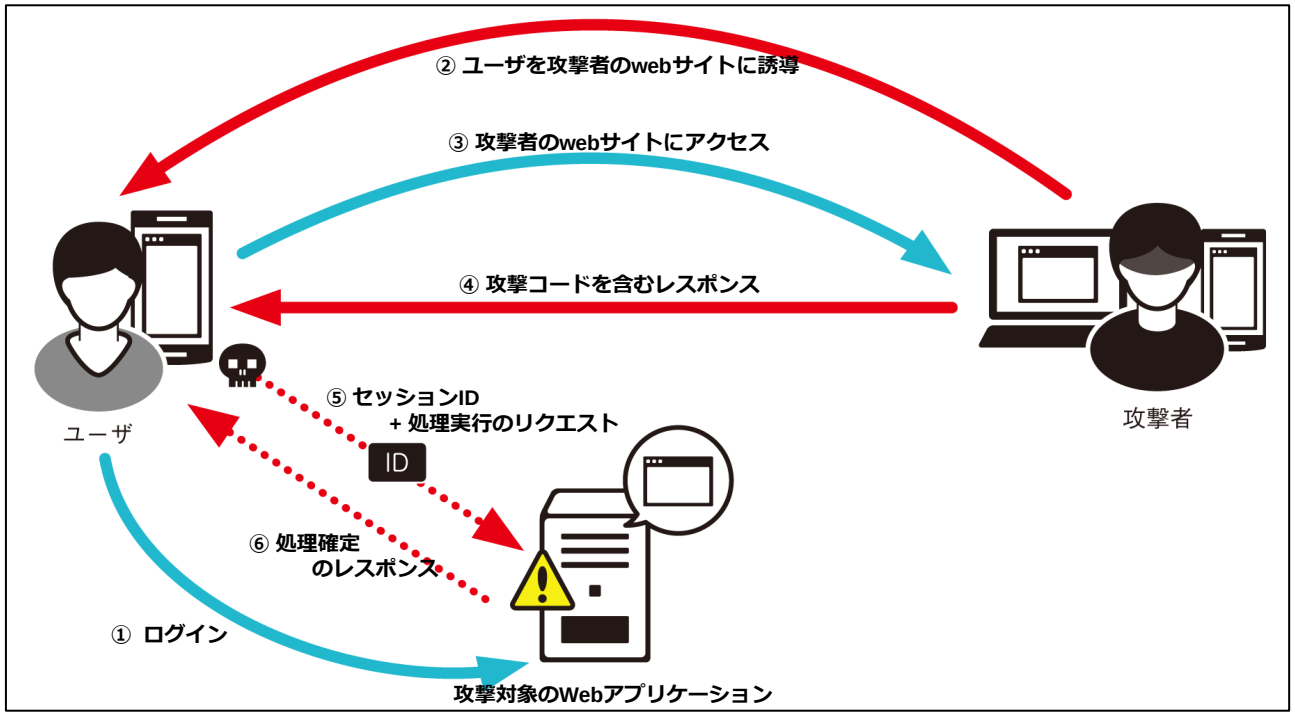
- 統合管理ツールとして以下が実施可能
  - ハードウエア、ソフトウエアのインベントリ管理
  - ソフトウェアのインストールやアップデート
  - 設定ファイルの管理と展開
  - システムのモニタリング

脆弱性の概要

- Spacewalkには、クロスサイトリクエストフォージェリの脆弱性が存在する。
- 脆弱性を悪用されることで、被害者が意図しない操作を実行させられる可能性がある。
- Spacewalkではシステムの管理機能を提供しているため、被害者が意図しない設定変更等が行われてしまう。

クロスサイトリクエストフォージェリとは

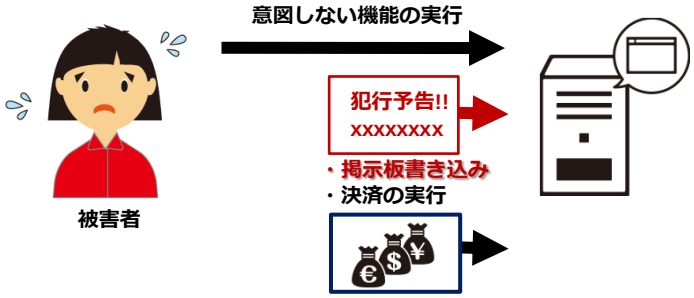
- 攻撃者の罠サイトにアクセスさせ、ログインしているWebサービスの機能を、意図せず実行させる攻撃。



脆弱性が悪用された場合のリスク

ユーザの意図しない機能の実行

- Webサイトが提供している機能により、リスクは異なる
- 例えば、掲示板に投稿する機能があった場合、攻撃者によって意図しない投稿をさせられるかも
- 商品を購入する機能があった場合、攻撃者によって意図しない商品購入をさせられるかも



Spacewalk の処理フロー

アカウント停止機能を実行する場合の処理フローを解説する。
画面遷移にしたがってユーザが操作を行い、処理が行われる。

アカウント停止機能を実行する際のSpacewalkの処理フロー

① ユーザがSpacewalkにログインする。

② ユーザがアカウント停止機能を実行しリクエストが送信される。

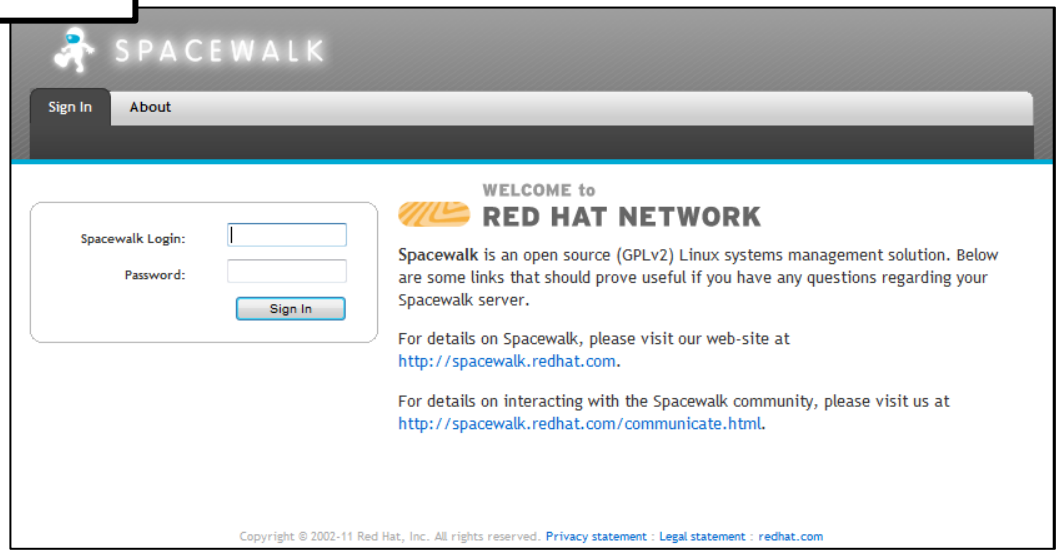
③ Spacewalkがリクエストを受信し、処理を実行する。

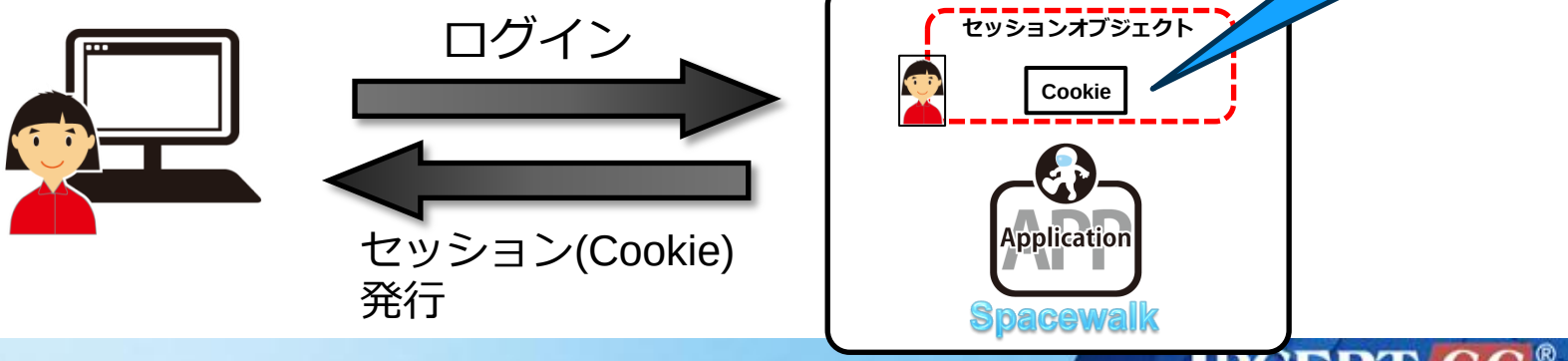
④ 結果を含むレスポンスがユーザへ送信される。

① ユーザがSpacewalkにログインする

ユーザがログインし、セッションを取得する。(Cookieを発行する)

ログイン画面

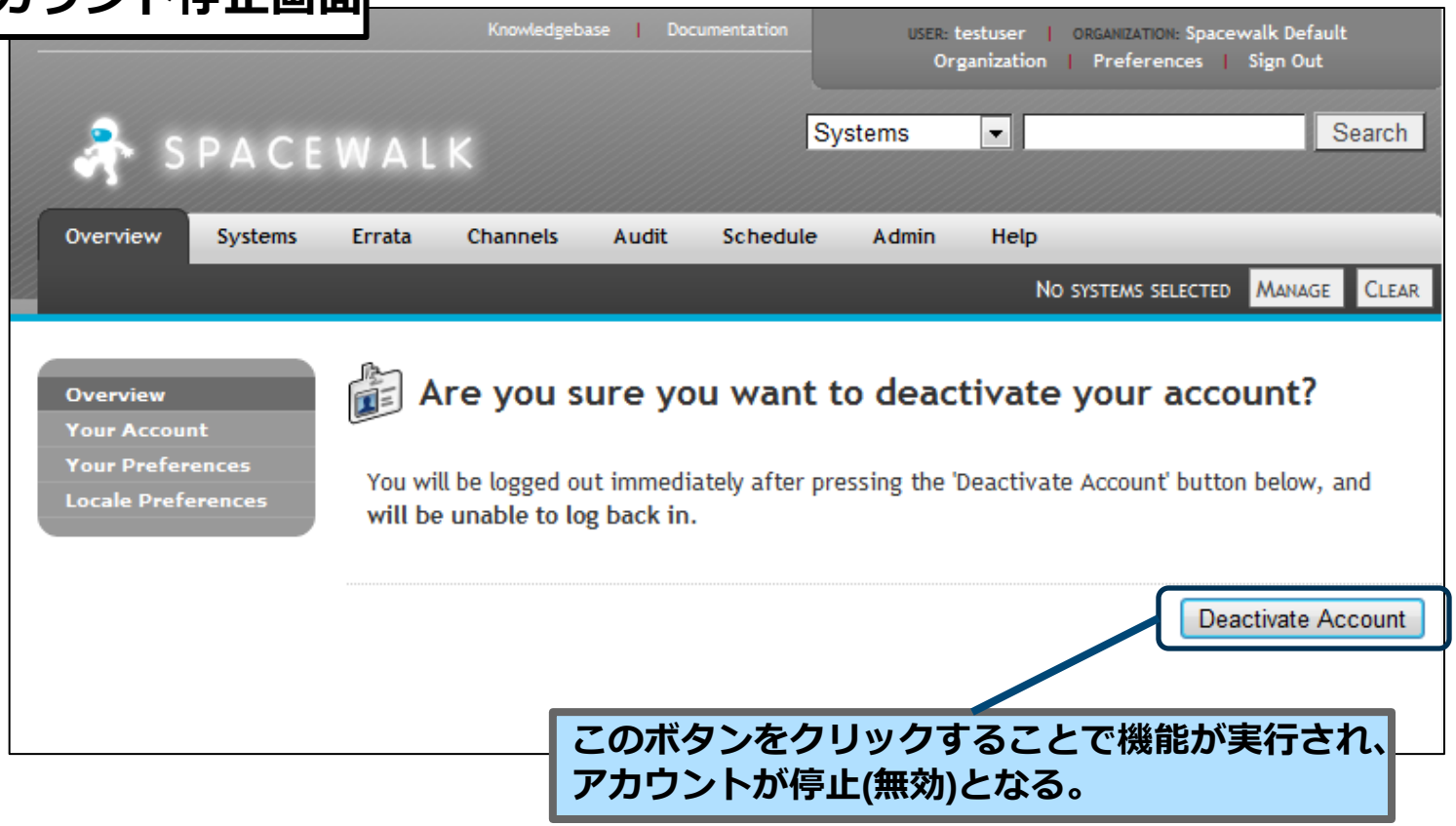




② ユーザがアカウント停止機能を実行しリクエストが送信される

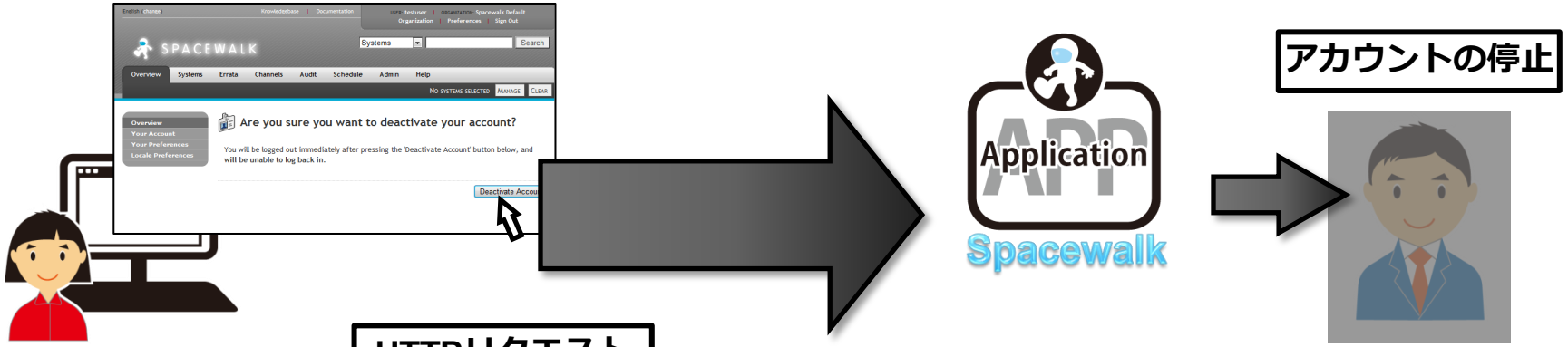
管理画面にログインし、アカウント停止画面にアクセスする。

アカウント停止画面



③Spacewalkがリクエストを受信し、処理を実行する

Spacewalkはリクエストを受信して、アカウント停止機能を実行する。



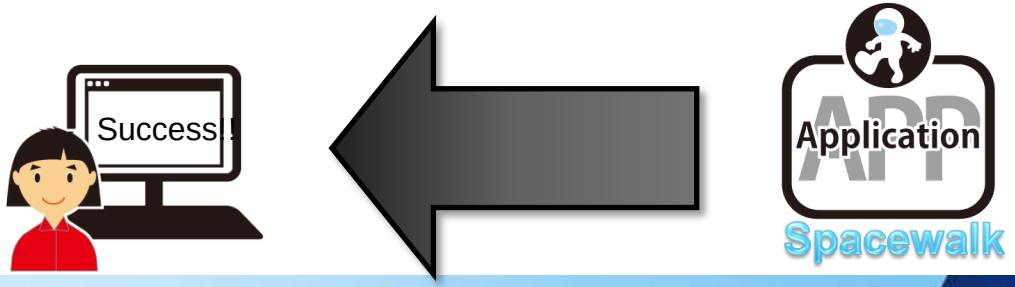
HTTPリクエスト

```
POST /rhn/account/AccountDeactivationConfirm.do HTTP/1.1
:
Host: www.example.com
Content-Length: 14
Cookie: JSESSIONID=81099D8047CF94FEA3FB447C3C24AF98;
 pxt-session-cookie=29x6ad3ed33446e2669c5e60a4b3080713a

submitted=true
```

④ 結果を含むレスポンスがユーザへ送信される

アカウント停止処理が終了し、結果をレスポンスとしてユーザへ送信する。

アカウントが無効となり、ログイン画面に戻る。



機能実行時　詳細

アカウント停止機能実行時の処理フローにおける、②の処理を詳しく見てみよう。

アカウント停止機能を実行する際のSpacewalkの処理フロー

① ユーザがSpacewalkにログインする

② ユーザがアカウント停止機能を実行しリクエストが送信される

③ Spacewalkがリクエストを受信し、処理を実行する

④ 結果を含むレスポンスがユーザへ送信される

機能実行時 詳細
② ユーザがアカウント停止機能を実行しリクエストが送信される

アカウント停止画面のForm要素（HTML）と、機能実行時のHTTPリクエスト

HTMLソース(抜粋)

```
        You will be logged out immediately after pressing the ‘Deactivate Account’ button below, and <b>will be
unable to log back in</b>.
      </p>
   </div>

   <hr />
   <form method="POST" name="rhn_list" action="/rhn/account/AccountDeactivationConfirm.do">
   <div align="right">
   <input type="submit" value="Deactivate Account" />
   </div>
   <input type="hidden" name="submitted" value="true" />
   </form>
```

「Deactivate Account」のクリックで送信されるForm要素

Deactivate ボタンを押したときに送信されるHTTPリクエスト

```
POST /rhn/account/AccountDeactivationConfirm.do HTTP/1.1
.
Host: www.example.com
Content-Length: 14
Cookie: JSESSIONID=81099D8047CF94FFA3FB447C3C24AF98; pxt-session-
cookie=29x6ad3ed33446e2669c5e60a4b3080713a

submitted=true
```

送信されるデータにForm要素固有の値は含まれておらず、毎回同じデータとなっている。

問題点

- Spacewalkがユーザからのリクエストを処理する際に、正しい画面遷移でリクエストが送信されてきたかを検証していなかった。

※ 参考

CWE(Common Weakness Enumeration)

「CWE-352 クロスサイトリクエストフォージェリ」

http://cwe.mitre.org/data/definitions/352.html

攻撃実行時の処理フロー

通常の処理フロー

アカウント停止機能を実行する際の処理フロー

① ユーザがSpacewalkにログインする
② ユーザがアカウント停止機能を実行しリクエストが送信される
③ Spacewalkがリクエストを受信し、処理を実行する
④ 結果を含むレスポンスがユーザへ送信される

攻撃実行時は、②の前にユーザが攻撃者サイトへ誘導され、以下のようなフローとなる。

攻撃実行時の処理フロー

アカウント停止機能を実行する際のSpacewalkの処理フロー

① ユーザがSpacewalkにログインする
② -1 ユーザが何らかの方法で攻撃者のサイトに誘導され、攻撃コードを実行するコンテンツにアクセスする
② -2 アカウント停止機能を実行するためのリクエストがユーザのブラウザからSpacewalkに対して送信される
③ Spacewalkがリクエストを受信し、処理を実行する
④ 結果を含むレスポンスがユーザへ送信される

誘導先の攻撃者サイト上のコンテンツ(攻撃コード)

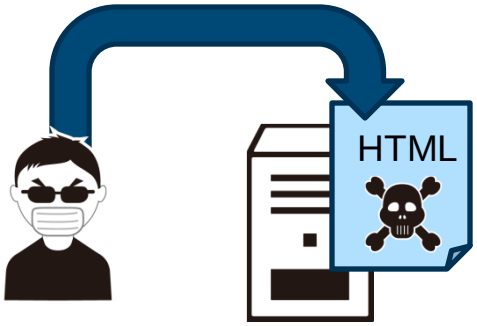
攻撃者は以下のようなコンテンツを用意し、Spacewalkにログイン中のユーザを誘導する。

攻撃コードのHTTPリクエストを送信するためのコンテンツ(HTML)

```
<html>
<body onload="document.atkform.submit()">
  <form method="POST" name="atkform"
        action="https://www.spacewalkserver.com/rhn/account/AccountDeactivationConfirm.do">
  <input type="hidden" name="submitted" value="true" />
  </form>
</body>
</html>
```

コンテンツにアクセスすると以下のForm要素が自動送信される。

アカウント停止機能を実行するForm要素



攻撃コードから送信されるリクエスト

コンテンツにアクセスすると送信されるHTTPリクエスト

```
POST /rhn/account/AccountDeactivationConfirm.do HTTP/1.1
:
Host: www.victim.com
Content-Length: 14
Cookie: JSESSIONID=81099D8047CF94FEA3FB447C3C24AF98;
 pxt-session-cookie=29x6ad3ed33446e2669c5e60a4b3080713a
submitted=true
```

①

■攻撃コードのポイント

- ユーザはSpacewalkにログイン中であるため、Cookieヘッダの値はSpacewalkから発行されたCookieが自動的に送信される。(①部分)
- このリクエストを受信したSpacewalkは、ログイン中のユーザーからのリクエストとして処理する。
- 通常の処理で送信されるリクエストと全く内容が一緒であり、Spacewalkにはリクエストが正常なものか(サイト内の正常な遷移で送信されたものかどうか)区別できない。

通常時/攻撃実行時のフロー比較

通常の処理フロー

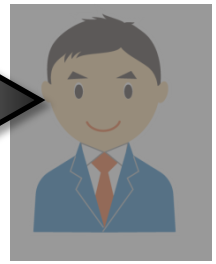
HTTPリクエスト

```
POST /rhn/account/AccountDeactivationConfirm.do HTTP/1.1
:
Host: www.example.com
Content-Length: 14
Cookie: JSESSIONID=81099D8047CF94FEA3FB447C3C24AF98;
 pxt-session-cookie=29x6ad3ed33446e2669c5e60a4b3080713a

submitted=true
```

Spacewalk

Application

機能の実行
アカウントの停止



攻撃者のサイトへ誘導された際の処理フロー

HTTPリクエスト

```
POST /rhn/account/AccountDeactivationConfirm.do HTTP/1.1
:
Host: www.example.com
Content-Length: 14
Cookie: JSESSIONID=81099D8047CF94FEA3FB447C3C24AF98;
 pxt-session-cookie=29x6ad3ed33446e2669c5e60a4b3080713a

submitted=true
```

アプリケーションにとっては全く同じリクエストであり、区別がつかない!!

対策

- 機能実行の際は、正しい画面遷移からの実行であることを確認するべき。
- 一般的なCSRF対策:
  - トークンを生成し、セッションに格納する
  - 同時にリクエストのForm要素にもトークンを含める
  - 機能実行時にセッションとリクエストのトークンが一致することを検証する
- 攻撃者がその値を予測したり総当たりで探索したりする可能性をふまえ、トークンの値は乱数を使って生成し、十分な長さを持った値にする必要がある。

修正版コード

脆弱性はバージョン1.2.39-85にて修正が適用されている。

アカウント停止機能を実行する際のSpacewalkの処理フロー

① ユーザがSpacewalkにログインする。

② 前処理　ユーザがアカウント停止画面にアクセス時に、Spacewalkはトークンを発行しForm要素に埋め込む。さらにセッション変数にトークンを格納する

② ユーザがアカウント停止機能を実行しリクエストが送信される。

②後処理　Spacewalkは送信されてきたトークンとセッション変数に格納されているトークンが同一であることを確認する。`

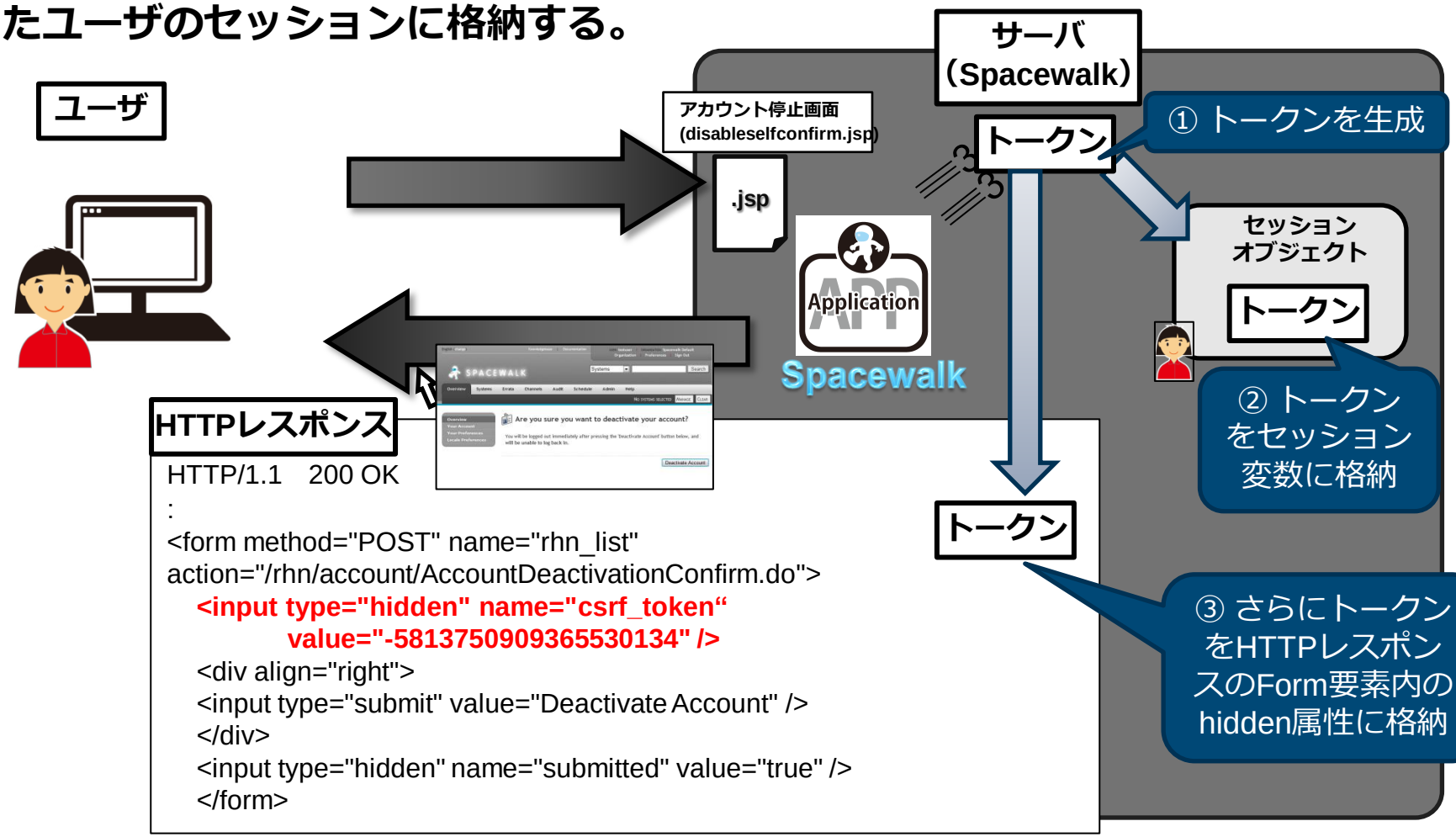
③ Spacewalkがリクエストを受信し、処理を実行する。

④ 結果を含むレスポンスがユーザへ送信される。

「②前処理」と「②後処理」が追加されている。

修正版コード
② Form要素へのトークン発行とセッション変数への格納

ユーザがアカウント停止画面にアクセスした際に、トークンを生成しアクセスしたユーザのセッションに格納する。



修正前/修正後のソース比較
② Form要素へのトークン発行とセッション変数への格納

disableselfconfirm.jsp (修正前)

```
<form method="post" name="rhn_list" action="/rhn/account/AccountDeactivationSubmit.do">
    <div align="right">
    <html:submit>
        <bean:message key="disableself.jsp.deactivate"/>
    </html:submit>
    </div>
    </form>
```

disableselfconfirm.jsp (修正後)

```
<form method="post" name="rhn_list" action="/rhn/account/AccountDeactivationSubmit.do">
    <rhn:csrf />
    <div align="right">
    <html:submit>
        <bean:message key="disableself.jsp.deactivate"/>
    </html:submit>
    </div>
    </form>
```

カスタムタグ rhn:csrf が追加されている

修正版コード
② Form要素へのトークン発行とセッション変数への格納

disableselfconfirm.jsp **<rhn:csrf />**

カスタムタグによりCsrfTagクラスのdoStartTagメソッドが呼び出される
(JSPの機能)

CsrfTag.java

```
public class CsrfTag extends HiddenTag {
 :
public int doStartTag() throws JspException {
    HttpServletRequest request = (HttpServletRequest) pageContext.getRequest();
    HttpSession session = request.getSession(true);

    this.setProperty("csrf_token");
    this.setValue(CSRFTokenValidator.getToken(session));

    super.doStartTag();
    return SKIP_BODY;
}
```

セッションを取得

トークンを生成し、
セッションに格納する

修正版コード
② Form要素へのトークン発行とセッション変数への格納

【参考】 CSRFTokenValidator クラスのgetTokenメソッド

```
public final class CSRFTokenValidator {
 :
   public static String getToken(HttpSession session) {
       String tokenValue = (String) session.getAttribute(TOKEN_KEY);
       if (tokenValue == null) {
           // Create new token if necessary
           tokenValue = createNewToken(DEFAULT_ALGORITHM);
           session.setAttribute(TOKEN_KEY, tokenValue);
       }
       return tokenValue;
   }
```

セッションへの格納

トークンの生成

修正版コード
② Form要素へのトークン発行とセッション変数への格納

disableselfconfirm.jsp **<rhn:csrf />**

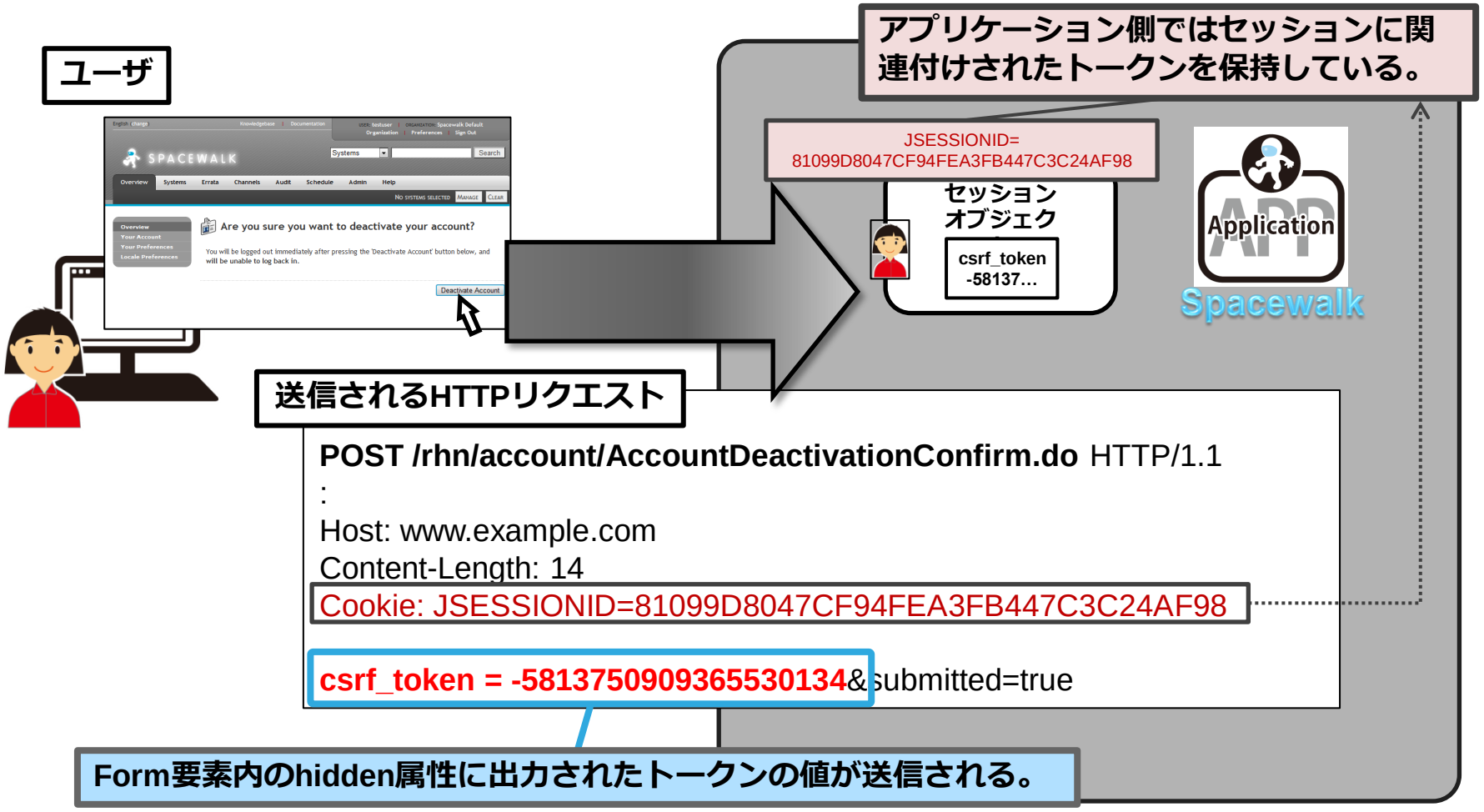
CsrfTagクラスはHiddenTagクラスを継承しており、setPropertyメソッドやsetValueメソッドで指定された値はForm要素内のhidden属性で出力される。

CsrfTag.java

```
public class CsrfTag extends HiddenTag {
 :
public int doStartTag() throws JspException {
    HttpServletRequest request = (HttpServletRequest) pageContext.getRequest();
    HttpSession session = request.getSession(true);

    this.setProperty("csrf_token");
    this.setValue(CSRFTokenValidator.getToken(session));

    super.doStartTag();
    return SKIP_BODY;
}
```

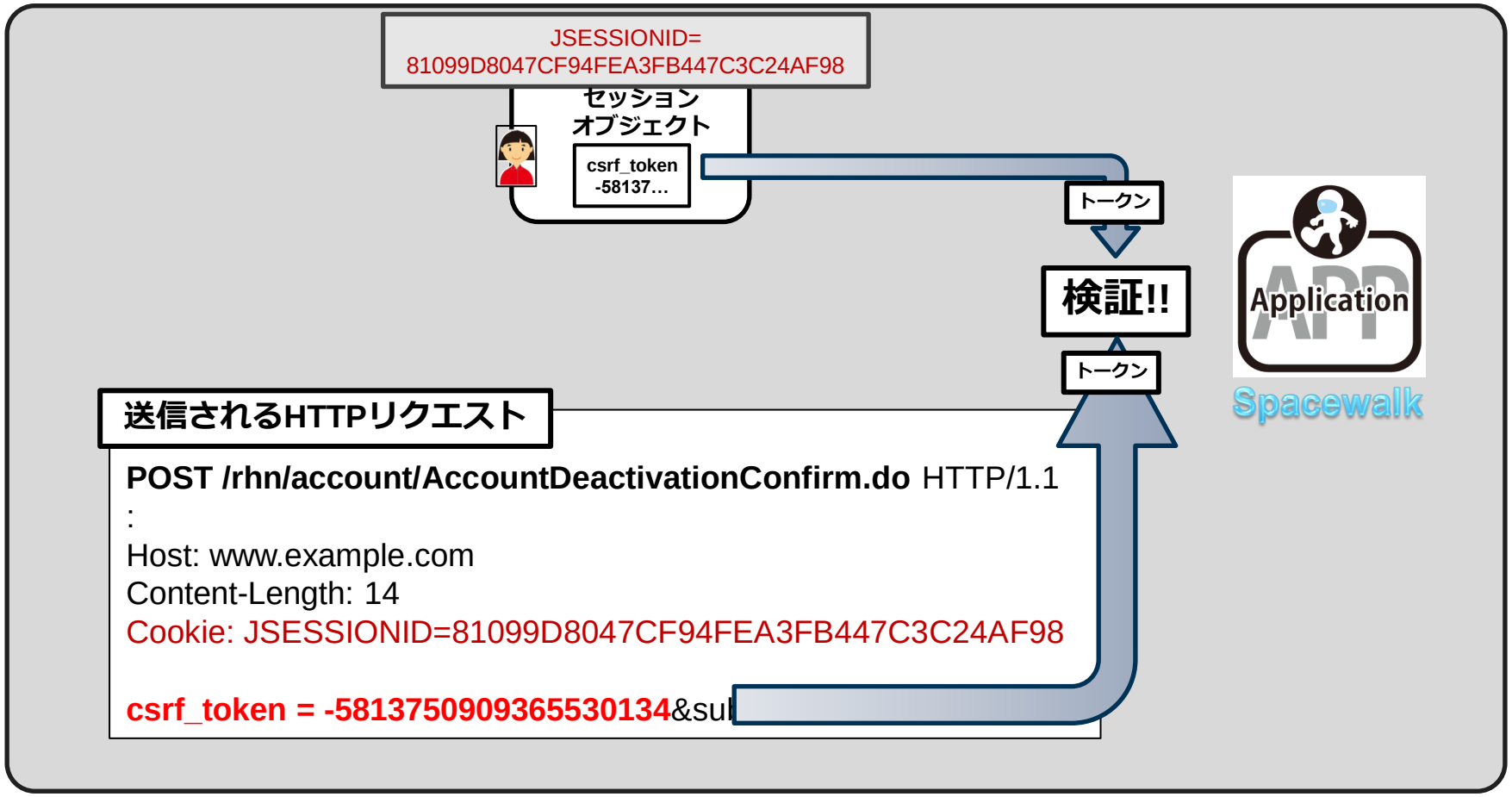
プロパティをセットすることでHTMLのForm要素内に以下のような形で出力される。
<input type="hidden" name="csrf_token" value="*<getToken()>*で作成されたトークン>" />

修正版コード
③ ユーザがアカウント停止機能を実行しリクエストが送信される



修正版コード
④ アプリケーションによるトークンの検証

Spacewalkは送信されてきたリクエスト内のトークンとセッション内のトークンが同じものであるかを検証する。



修正版コード
④ アプリケーションによるトークンの検証

リクエスト受信時にCSRFTokenValidatorクラスのvalidateメソッドが呼び出され、トークンの検証を行う。(本クラスは修正版コードに追加されたもの)

CSRFTokenValidator .java

```
public final class CSRFTokenValidator {
 :
public static void validate(HttpServletRequest request) throws CSRFTokenException {
        HttpSession session = request.getSession();

        if (session.getAttribute(TOKEN_KEY) == null) {
            throw new CSRFTokenException("Session does not contain a CSRF security token");
        }

        if (request.getParameter(TOKEN_KEY) == null) {
            throw new CSRFTokenException("Request does not contain a CSRF security token");
        }

        if (!session.getAttribute(TOKEN_KEY).equals(request.getParameter(TOKEN_KEY))) {
            throw new CSRFTokenException("Validation of CSRF security token failed");
        }
    }
```

セッション変数とリクエストにトークンが含まれているかを検証

修正版コード
④ アプリケーションによるトークンの検証

リクエスト受信時にCSRFTokenValidatorクラスのvalidateメソッドが呼び出され、トークンの検証を行う。(本クラスは修正版コードに追加されたもの)

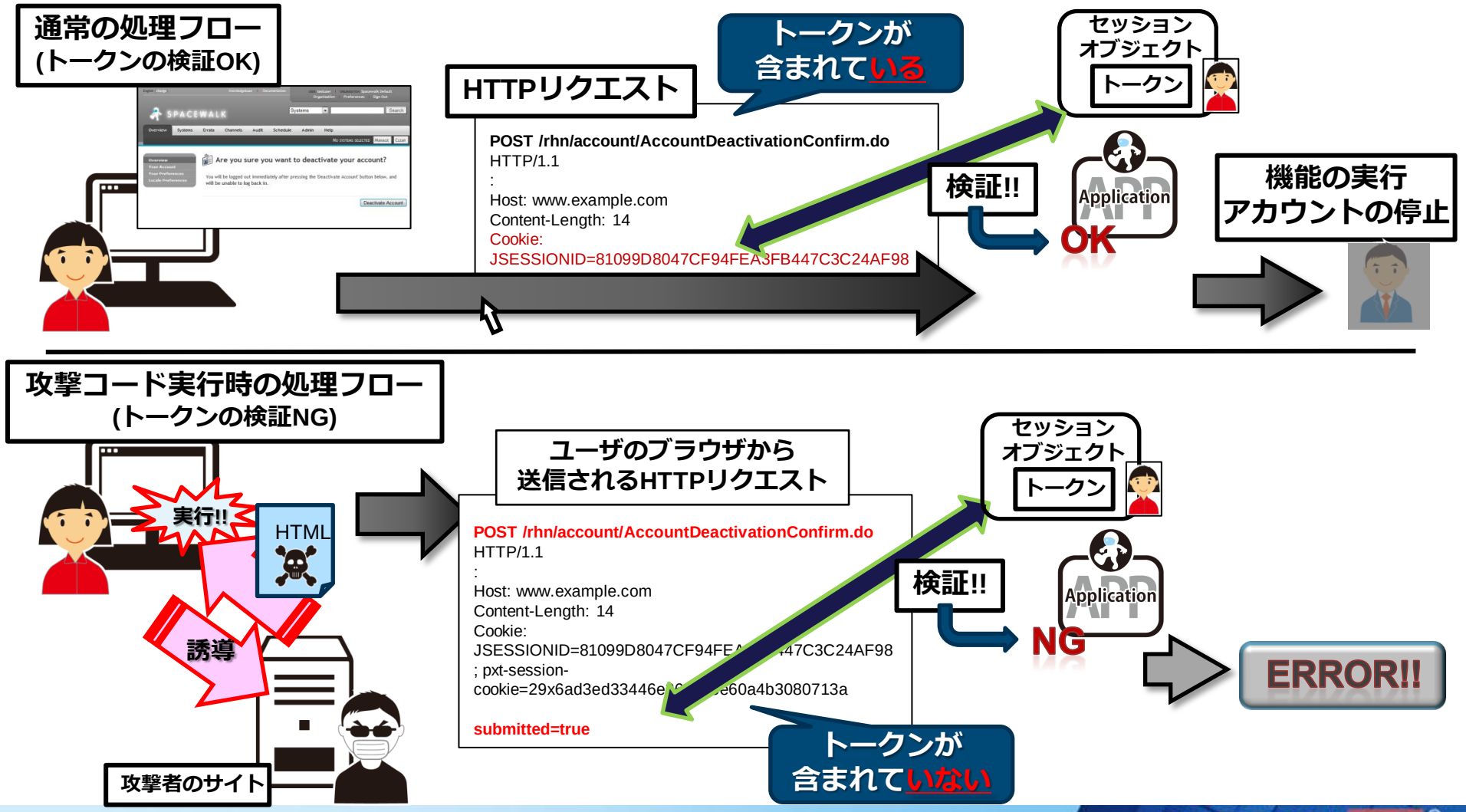
CSRFTokenValidator .java

```
public final class CSRFTokenValidator {
 :
public static void validate(HttpServletRequest request) throws CSRFTokenException {
        HttpSession session = request.getSession();

        if (session.getAttribute(TOKEN_KEY) == null) {
            throw new CSRFTokenException("Session does not contain a CSRF security token");
        }

        if (request.getParameter(TOKEN_KEY) == null) {
            throw new CSRFTokenException("Request does not contain a CSRF security token");
        }

        if (!session.getAttribute(TOKEN_KEY).equals(request.getParameter(TOKEN_KEY))) {
            throw new CSRFTokenException("Validation of CSRF security token failed");
        }
    }
```

セッション変数とリクエストに含まれるトークンが同一であるかを検証

修正版コード
⑤ アプリケーションがリクエストを受信し、処理を実行する

トークンの検証結果に応じて、機能実行とエラー処理のどちらかを行う



まとめ

■クロスサイトリクエストフォージェリ

Spacewalk が用意したフォームからの入力と攻撃者が用意したフォームからの入力を区別していなかった

■対策: トークンを使用することで、不正なフォームからの入力を検出できるようにする

トークン使用時には以下の点を検討しておく必要がある

- 攻撃者に値を予測されることを防ぐ: 乱数を使って生成
- 総当たりで探索されることを防ぐ: 長い文字列長、大きな数値など
- ユーザと関連付ける
- トークンの有効期間を明確にする: 処理を受け付けたら無効にする, リクエストが来なかったら60秒で無効にする、など

参考文献

■ OWASP CSRFGuard Project

https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

■ IPA ISEC セキュア・プログラミング講座：
Webアプリケーション編
第4章 セッション対策：リクエスト強要（CSRF）対策

https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/301.html

IPA 独立行政法人 情報処理推進機構
セキュア・プログラミング講座
SECURE PROGRAMMING COURSE

■安全なウェブサイトの作り方、IPA

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

著作権・引用や二次利用について

■本資料の著作権はJPCERT/CCに帰属します。

■本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URL の明示をお願いします。

記載例

引用元：一般社団法人JPCERTコーディネーションセンター
Java アプリケーション脆弱性事例解説資料
Spacewalk における CSRF の脆弱性
https://www.jpcert.or.jp/securecoding/2012/No.08_Spacewalk.pdf

■本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(office@jpcert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ
JPCERTコーディネーションセンター
広報担当
E-mail：office@jpcert.or.jp

本資料の技術的な内容に関するお問い合わせ
JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail：secure-coding@jpcert.or.jp