電子署名者 : Japan Computer Emergency Response Team Coordination Center
DN : c=JP, st=Tokyo, l=Chiyoda-ku, email=office@jpcert.or.jp, o=Japan Computer Emergency Response Team Coordination Center, cn=Japan Computer Emergency Response Team Coordination Center
日付 : 2013.09.27 09:08:12 +09'00'

「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

Copyright©2013 JPCERT/CC All rights reserved.

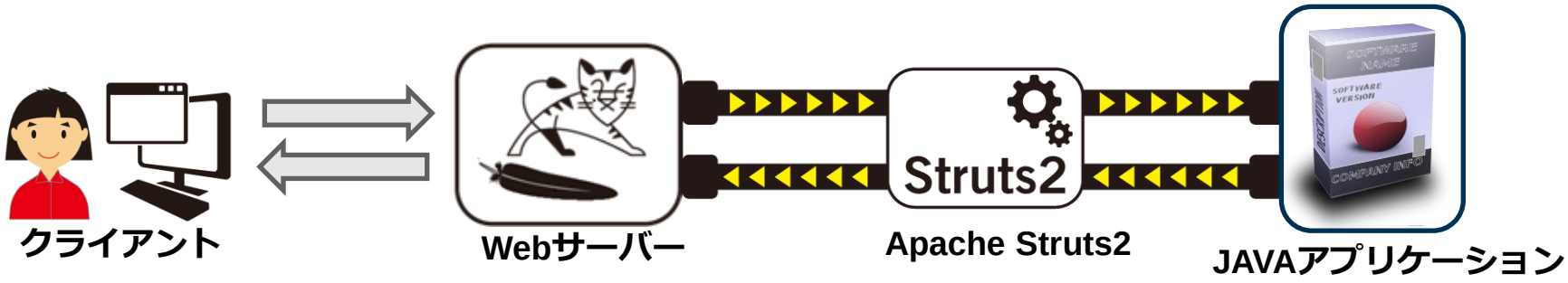
Apache Struts 2 における任意の Java メソッド実行の脆弱性

CVE-2012-0838

JVNDB-2012-000012

Apache Struts 2とは

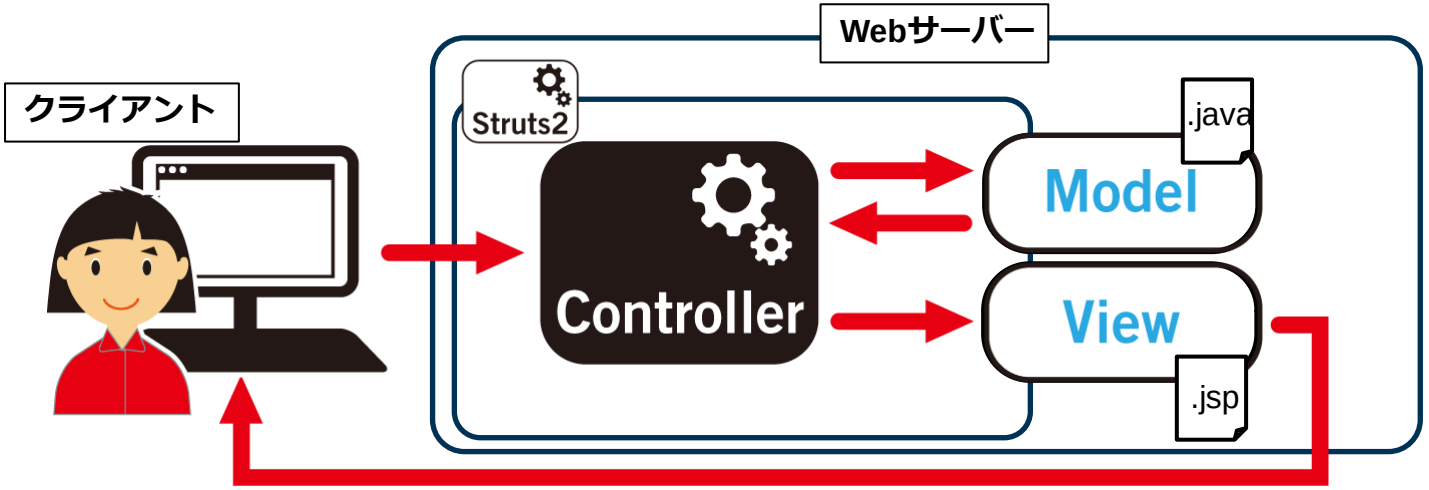
- Java のWebアプリケーションを開発するためのオープンソースのフレームワーク
- アプリケーションの開発効率やメンテナンス性を向上させることを目的としている



Copyright©2013 JPCERT/CC All rights reserved.

Apache Struts 2とは

- Model-View-Controller (MVC)アーキテクチャを採用
- アプリケーションの開発者はModelとViewを実装する。



Model: ビジネスロジックの実装 (javaで実装)

View: 画面デザイン (jspで実装)

Controller: クライアントからの入力イベントを処理し、ModelやViewに振り分ける(Struts2で実装)

Copyright©2013 JPCERT/CC All rights reserved.

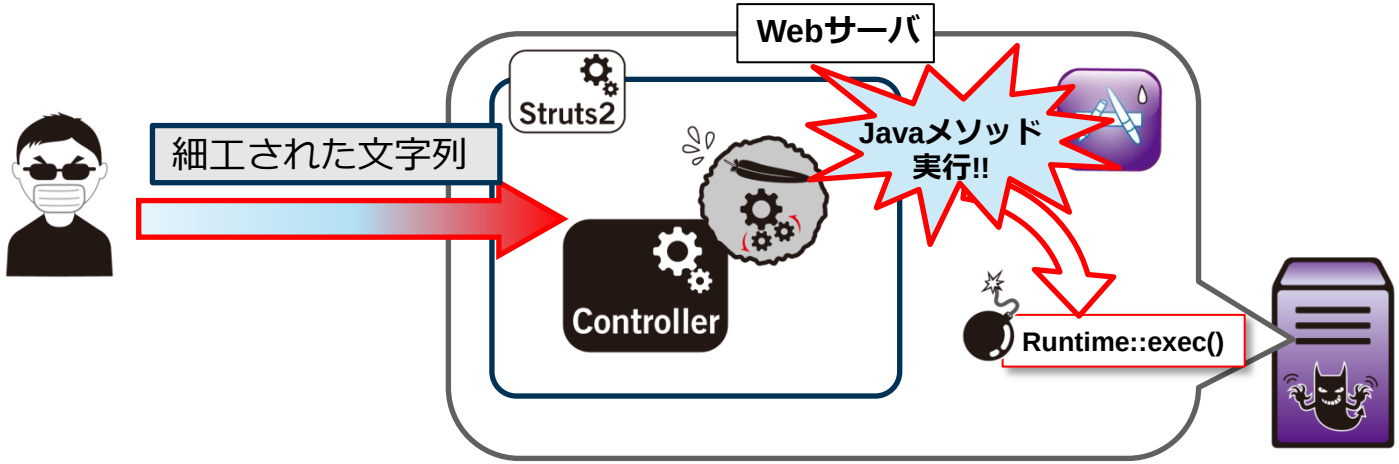
脆弱性の概要

- クライアントから送信されてきた文字列の処理で発生するエラーの取り扱いに不備が存在。
- 具体的には、数値型の変数に対して文字列が送信されてきた際に発生する変換エラー処理。
- 送信する文字列を細工することで、任意のJavaメソッドが実行可能となる。

Copyright©2013 JPCERT/CC All rights reserved.

脆弱性が悪用された場合のリスク

- 任意のJavaメソッドが実行できるため、Runtime::exec メソッドを使えばOSコマンドを実行することが可能
- アプリケーションの外部から、アプリケーションが稼働するサーバ上で任意のコマンドが実行可能となり、サーバを乗っ取られる可能性がある



Copyright©2013 JPCERT/CC All rights reserved.

Struts2 内部で変換エラーが発生した場合の処理

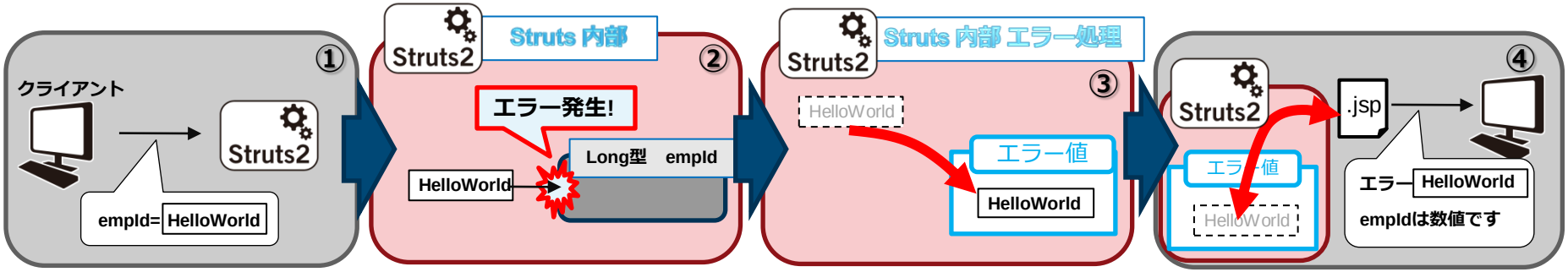
※Struts2付属のサンプルアプリ「showcase」を元に処理を解説する

① クライアントから、エラーの原因となる文字列を含むリクエストが送信される。

② Struts上で、文字列→数値に変換する際にエラーが発生し、エラー(例外)処理を行う。

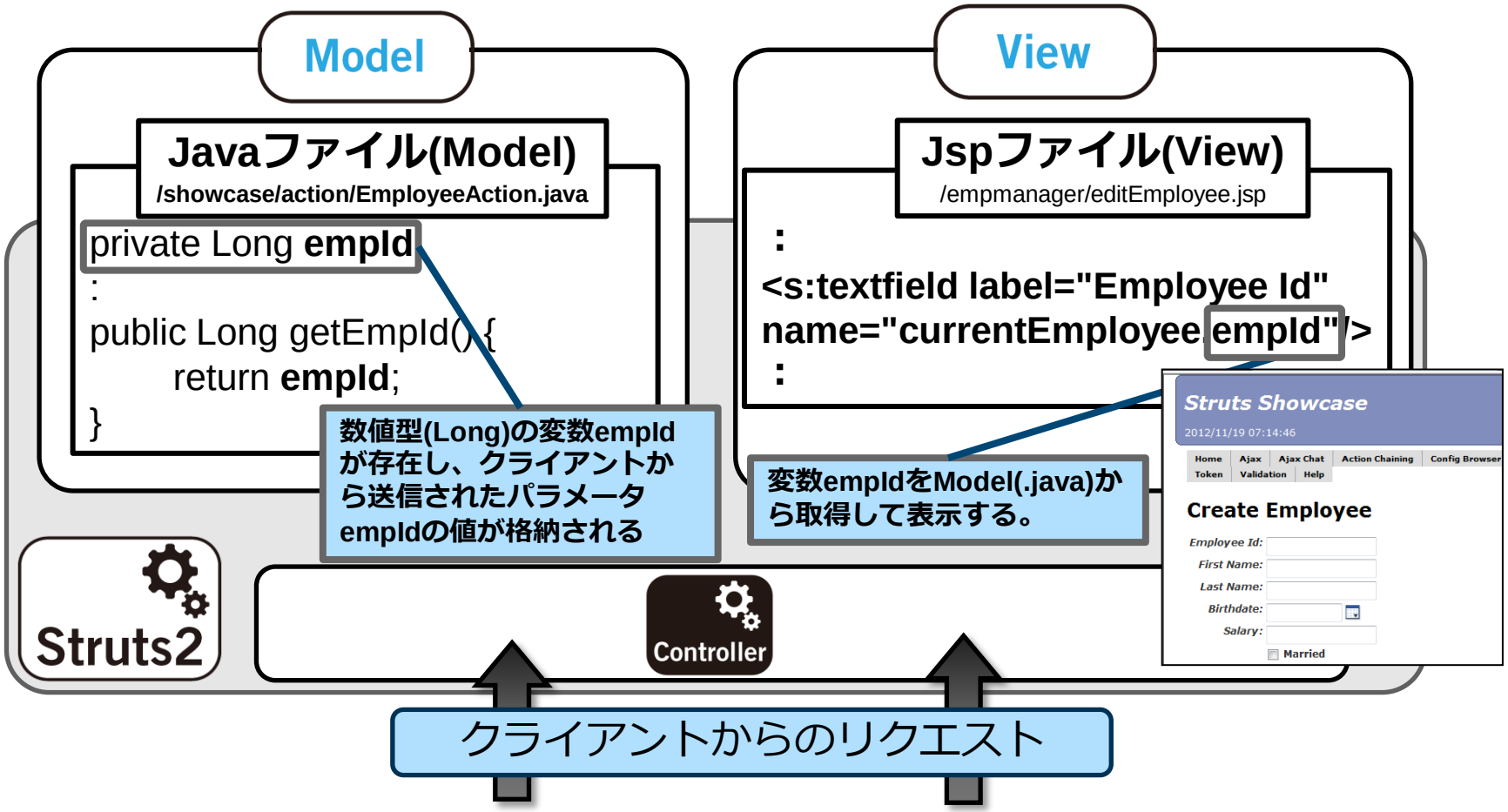
③ エラー処理で、変換エラーの原因となった文字列が保持される。

④ jsp側からエラーの原因となった文字列が取り出され、レスポンスの一部としてクライアントへ送信される。



Copyright©2013 JPCERT/CC All rights reserved.

サンプルアプリ「showcase」



数値型の変数**empId**に対して文字列を送信するとエラーが発生する。

Copyright©2013 JPCERT/CC All rights reserved.

①クライアントからエラーの原因となる文字列を含むリクエストが送信される

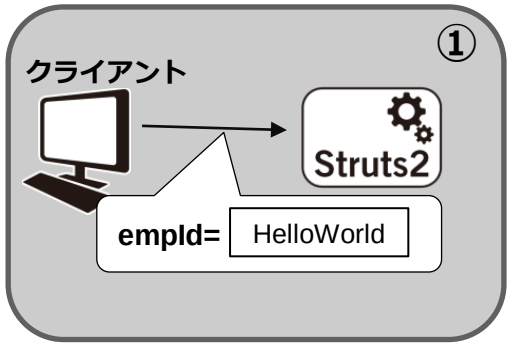
HTTPリクエスト

```
GET /?empId=HelloWorld HTTP/1.1
Host: localhost:8080
```

上記HTTPリクエストを送信するためのHTML

```
<form action='/' method='GET'>
:
<input type='text' name='empId' value='HelloWorld'>
:
</form>
```

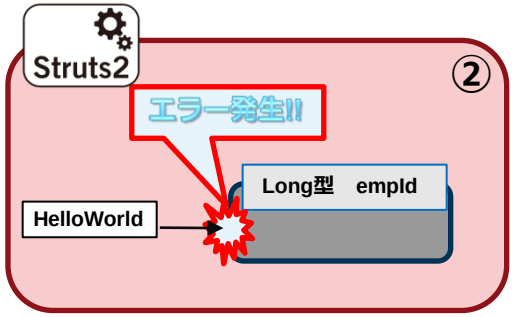
パラメータempIdの値として"HelloWorld"という文字列を送信する。(empIdはサーバ側では数値型として扱われることに注意。)



Copyright©2013 JPCERT/CC All rights reserved.

②文字列→数値に変換する際にエラー発生、エラー(例外)処理を行う

クライアントから送信されてきたパラメータempIdの値”HelloWorld”を、数値型変数empIdに対して挿入する際に変換エラーが発生し、エラー処理が行われる。



変換エラー処理はConversionErrorInterceptorクラスのinterceptメソッド内でエラー例外処理が行われる。

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
        :
        public String intercept(ActionInvocation invocation) throws Exception {
                :
```


Copyright©2013 JPCERT/CC All rights reserved.

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation) throws Exception {
        :
        HashMap<Object, Object> fakie = null;
        :
        String propertyName = (String) entry.getKey();
        Object value = entry.getValue();
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
```

currentEmployee.empId

エラーとなった変数名が
<クラス名.変数名>
という形式で取得され、変数
propertyNameに格納される。

Copyright©2013 JPCERT/CC All rights reserved.

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation) throws Exception {
        :
        HashMap<Object, Object> fakie = null;
        :
        String propertyName = (String) entry.getKey();
        Object value = entry.getValue();
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
```

“HelloWorld”

変数valueには変換エラーとなった文字列の値が代入される。

Copyright©2013 JPCERT/CC All rights reserved.

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation) throws Exception {
        :
        HashMap<Object, Object> fakie = null;
        :
        String propertyName = (String) entry.getKey();
        Object value = entry.getValue();
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
```

‘HelloWorld’

“HelloWorld”

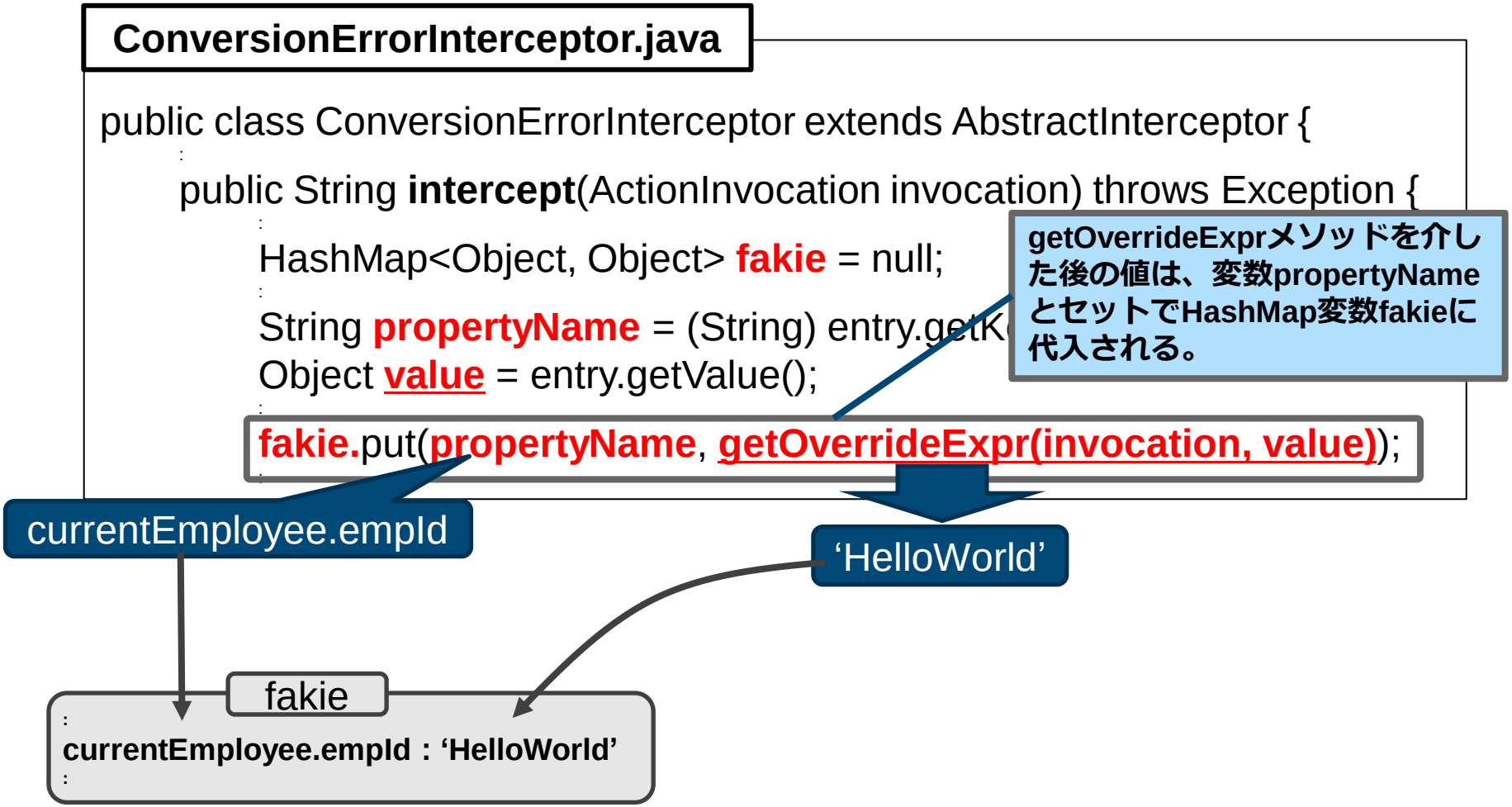
getOverrideExprメソッド

```
protected Object getOverrideExpr(ActionInvocation invocation, Object value) {
    return "'" + value + "'";
}
```

getOverrideExprメソッドでは第2引数を’（シングルクオート）で囲って返却する処理を行う

Copyright©2013 JPCERT/CC All rights reserved.

③エラー処理で、変換エラーの原因となった文字列が保持される



Copyright©2013 JPCERT/CC All rights reserved.

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

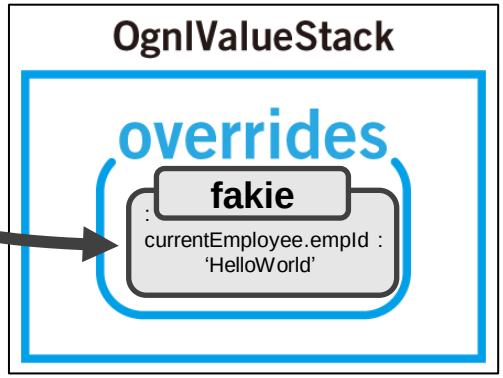
```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation) throws Exception {
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
        invocation.getStack().setExprOverrides(fakie);
```

fakie

```
:
currentEmployee.empId : 'HelloWorld'
:
```

OgnlValueStack.java

```
public void setExprOverrides(Map overrides) {
    if (this.overrides == null) {
        this.overrides = overrides;
    }
```



エラー値を含む変数fakieはOgnlValueStackクラスのインスタンス変数overridesとして保持される。

Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する。

Jspに記述されたコード(OGNL式)にしたがって、変数empIdの値が取得される。

Jspファイル(View)
/empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

- OGNLとは

⇒Object-Graph Navigation Language (OGNL)

- OGNLはJavaのクラス(Model)の変数を設定/取得するための言語
- サンプルコードのjspに記載されたOGNL式は、変数 **currentEmployee.empId**の値を取得して表示をする式

Point! 下記のような表記で直接**Javaメソッドを呼び出すことも可能**

```
<property name="roots">@java.io.File@listRoots()</property>
```

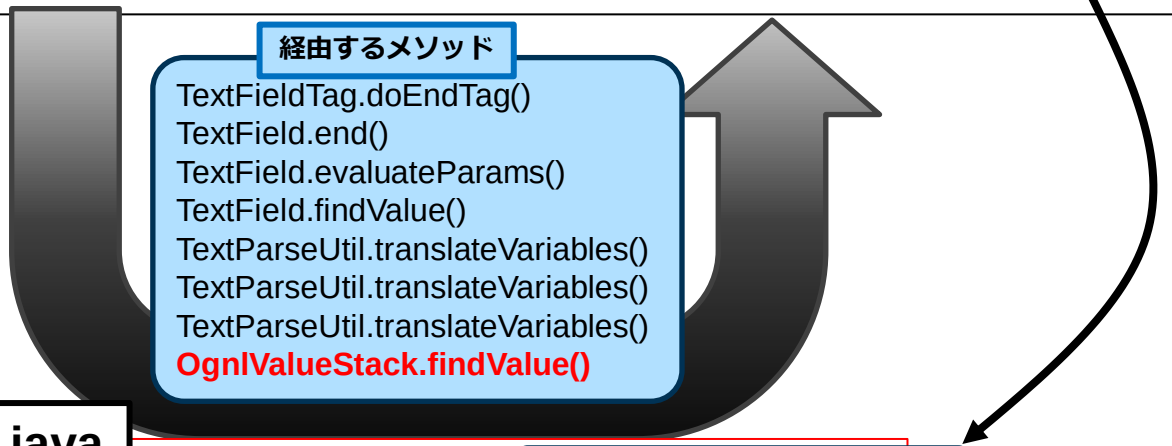

Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する。

Jspに記述されたOGNL式にしたがって、OgnlValueStack.findValueメソッドを経由して変数currentEmployee.empId の値(エラーとなった文字列)が取得される。

Jspファイル(View) /empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```



currentEmployee.empId

OgnlValueStack.java

```
public Object findValue(String expr, Class asType) {
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
        expr = (String) overrides.get(expr);
        :
        Object value = OgnlUtil.getValue(expr, context, root, asType);
        :
    }
```


Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する。

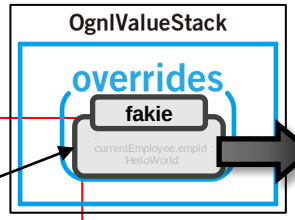
Jspに記述されたOGNL式にしたがって、OgnlValueStack.findValueメソッドを経由して変数currentEmployee.empId の値(エラーとなった文字列)が取得される。

Jspファイル(View) /empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

経由するメソッド

TextFieldTag.doEndTag()
TextField.end()
TextField.evaluateParams()
TextField.findValue()
TextParseUtil.translateVariables()
TextParseUtil.translateVariables()
TextParseUtil.translateVariables()
OgnlValueStack.findValue()



'HelloWorld'

OgnlValueStack.java

```
public Object findValue(String expr, Class asTy
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
        expr = (String) overrides.get(expr);
        :
        Object value = OgnlUtil.getValue(expr, context,
        :
    }
```

currentEmployee.empId

'HelloWorld'

前述の変数overridesから、変数exprとセットの値(エラーが発生する原因となった文字列)がとりだされる。

Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する。

Jspに記述されたOGNL式にしたがって、OgnlValueStack.findValueメソッドを経由して変数currentEmployee.empId の値(エラーとなった文字列)が取得される。

Jspファイル(View) /empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

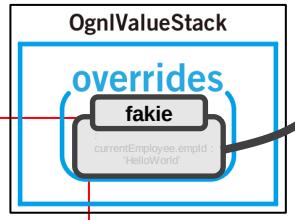
文字列'HelloWorld'はOGNL式として意味を持たないため、'HelloWorld'のまま

経由するメソッド

TextFieldTag.doEndTag()
TextField.end()
TextField.evaluateParams()
TextField.findValue()
TextParseUtil.translateVariables()
TextParseUtil.translateVariables()
TextParseUtil.translateVariables()
OgnlValueStack.findValue()

'HelloWorld'

OGNL



OgnlValueStack.java

```
public Object findValue(String expr, Class asType) {
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
            expr = (String) overrides.get(expr);
            :
        Object value = OgnlUtil.getValue(expr, context, root, asType);
            :
    }
```

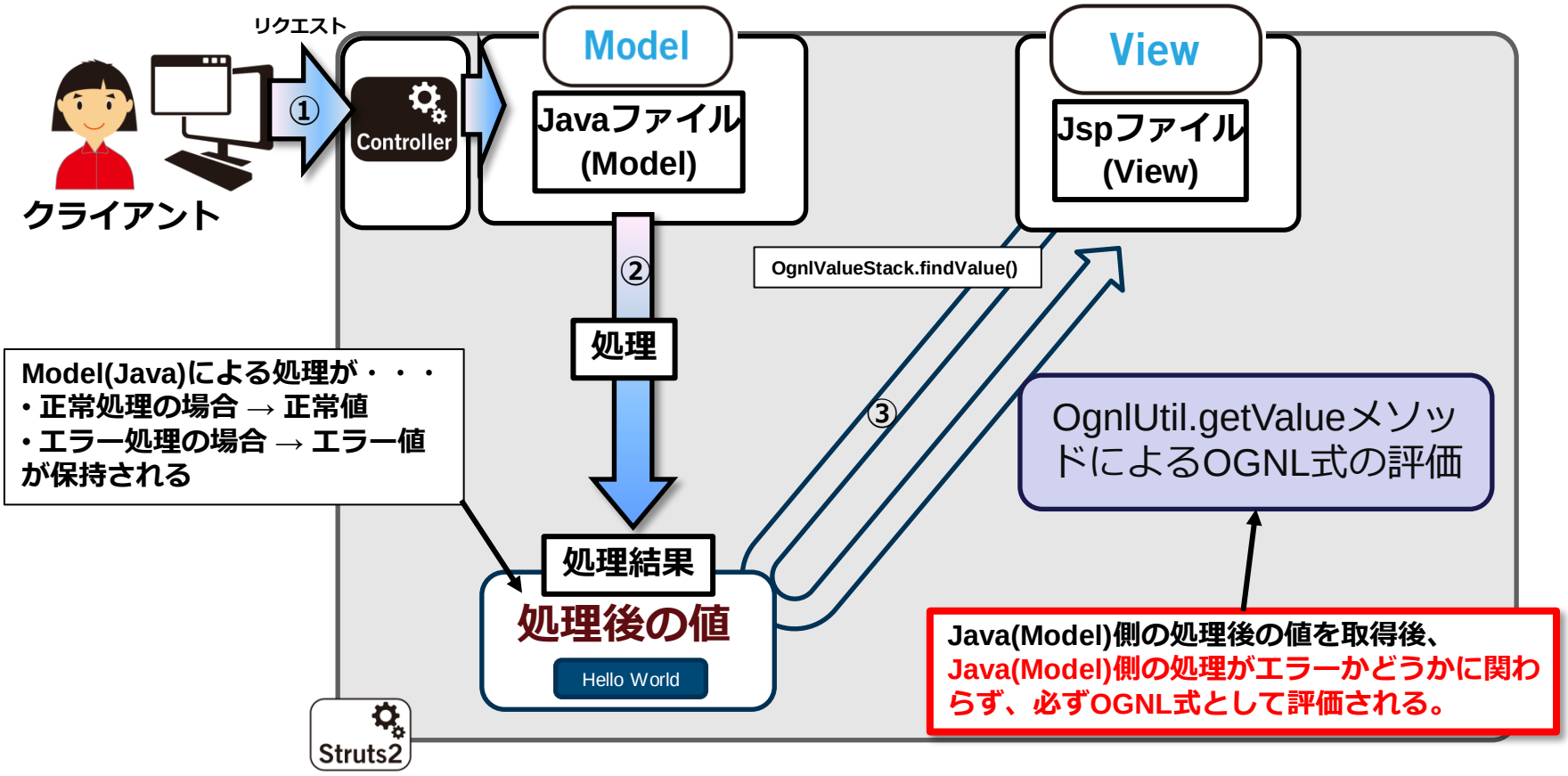
Point!

変数exprの値はOgnlUtil::getValueメソッドにてOGNL式として評価された後にクライアントに返却される。

Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する。

Jspに記述されたOGNL式では、OgnlValueStack.findValueメソッドによってJava(Model)の処理結果を取得後、必ずOgnlUtil.getValueメソッドによってOGNL式による評価が行われる。



Copyright©2013 JPCERT/CC All rights reserved.

④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する

レスポンスがクライアントに返される。
クライアント上ではエラーメッセージが表示される。

Copyright©2013 JPCERT/CC All rights reserved.

攻撃コード

攻撃コードのHTTPリクエスト

```
GET /?empId='%2B@java.lang.Runtime@getRuntime().exec(“notepad.exe")%2B'  HTTP/1.1
Host: localhost:8080
```

攻撃コードのHTTPリクエストを送信するためのHTML

```
<form action=‘/’ method=‘GET’>
:
<input type=‘hidden’ name=‘empId’
value="'+@java.lang.Runtime@getRuntime().exec(&quot;notepad.exe&quot;)+'">
:
</form>
```

■攻撃コードのポイント
パラメーターempIdに対し、JavaメソッドであるgetRuntime.execでコマンドを実行するOGNL式を指定している。

Copyright©2013 JPCERT/CC All rights reserved.

攻撃コードが実行された際の処理

変換エラーが発生した際の処理フロー

※Struts2に付属しているサンプルアプリ「showcase」を元に処理を解説する

① クライアントから、エラーの原因となる文字列を含むリクエストが送信される。

② Struts上で、文字列→数値に変換する際にエラーが発生し、エラー（例外）処理を行う。

③ エラー処理で、変換エラーの原因となった文字列が保持される。

④ Jsp側からエラーの原因となった文字列が取り出され、レスポンスの一部としてクライアントへ送信される。

攻撃コードが実行された際でも処理のフローは正常処理と全く変わらないが、④の処理内容で違いが発生する。

Copyright©2013 JPCERT/CC All rights reserved.

攻撃コード実行時
④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する

Jspの内容にしたがって、変数empIdの値が取得され、エラーの原因となった文字列が返されるが・・・

Jspファイル(View) /empmanager/editEmployee.jsp

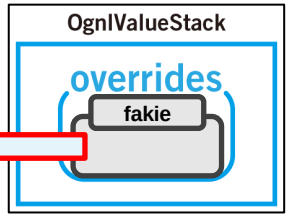
```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

OgnlValueStack.java

```
public Object findValue(String expr, Class asType) {
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
        expr = (String) overrides.get(expr);
        :
        Obj    value = OgnlUtil.getValue(expr, context, root, asType);
        :
    }
```

前述の変数overridesから、エラーが発生した原因となる文字列が変数exprにとりだされる。

'+@java.lang.Runtime@getRuntime().exec(“notepad.exe")+'



Copyright©2013 JPCERT/CC All rights reserved.

攻撃コード実行時
④Jsp側からエラーの文字列が取り出され、レスポンスとしてクライアントへ送信する

Jspの内容にしたがって、変数empIdの値が取得され、エラーの原因となった文字列が返されるが・・・

Jspファイル(View) /empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

OgnlValueStack.java

```
public Object findValue(String expr, Class asType) {
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
        expr = (String) overrides.get(expr);
        :
        Object value = OgnlUtil.getValue(expr, context, root, asType);
        :
    }
```

変数exprがOGNL式として評価される際にＪａｖａメソッドが実行されてしまう！！

'+@java.lang.Runtime@getRuntime().exec(“notepad.exe")+'

Copyright©2013 JPCERT/CC All rights reserved.

問題点

- 今回のアプリケーションにおける具体的な問題点

  引数をOGNL式として評価するOgnlUtil::getValueメソッドに対し、無害化していないデータを渡してしまっていた。

以下のコーディングガイドに違反している!!
「 IDS00-J. 信頼境界を越えて渡される信頼できないデータは無害化する」

- 問題点に対してどうすべきだったか

  OgnlUtil::getValueメソッドの引数として渡す前に、データがOGNL式として解釈されないように無害化する必要があった。

Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: バージョン2.2.3.1で修正

変換エラーが発生した際の処理フロー
※Struts2付属のサンプルアプリ「showcase」を元に処理を解説する

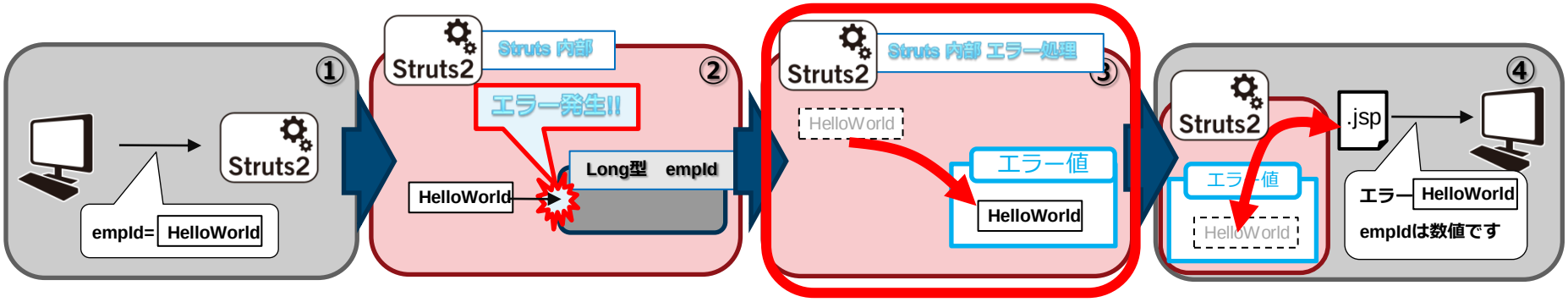
① クライアントから、エラーの原因となる文字列を含むリクエストが送信される。

② Struts上で、文字列→数値に変換する際にエラーが発生し、エラー（例外）処理を行う。

この処理のコードに修正が入っている

③ エラー処理で、変換エラーの原因となった文字列が保持される。

④ Jsp側からエラーの原因となった文字列が取り出され、レスポンスの一部としてクライアントへ送信される。



Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: 脆弱バージョンのおさらい

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation) throws Exception {
        :
        lashMap<Object, Object> fakie = null;
        :
        Object value = entry.getValue();
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
```

“HelloWorld”

変数valueには変換エラーとなった入力値が代入される。

Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: 脆弱バージョンのおさらい

③エラー処理で、変換エラーの原因となった文字列が保持される

ConversionErrorInterceptor.java

```
public class ConversionErrorInterceptor extends AbstractInterceptor {
    :
    public String intercept(ActionInvocation invocation)
        :
        HashMap<Object, Object> fakie = null;
        :
        Object value = entry.getValue();
        :
        fakie.put(propertyName, getOverrideExpr(invocation, value));
        :
```

変数valueは getOverrideExpr メソッドを介してHashMap変数fakieに代入される。

getOverrideExprメソッド

```
protected Object getOverrideExpr(ActionInvocation invocation, Object value) {
    return "'" + value + "'";
}
```

getOverrideExprメソッドでは第2引数を'(シングルクオート)で囲って返却する処理を行う

修正版コードではgetOverrideExprメソッドに修正が加えられている。

Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: getOverrideExprメソッドの修正内容

③エラー処理で、変換エラーの原因となった文字列が保持される

getOverrideExprメソッド（修正前）

```
protected Object getOverrideExpr(ActionInvocation invocation, Object value) {
        return "'" + value + "'";
}
```

getOverrideExprメソッドでは第2引数を'(シングルクオート)で囲って返却する処理を行う

StringEscapeUtils.escapeJavaメソッドを使ってエスケープ処理を行っている。

getOverrideExprメソッド（修正後）

```
protected Object getOverrideExpr(ActionInvocation invocation, Object value) {
        return "¥'" + StringEscapeUtils.escapeJava(String.valueOf(value)) + "¥'";
}
```


Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: StringEscapeUtils.escapeJavaメソッドによるエスケープ

- **StringEscapeUtils.escapeJava**メソッドによるエスケープ処理

エスケープ対象文字列	エスケープ処理後の文字列
¥b	¥¥b
¥n	¥¥n
¥t	¥¥t
¥f	¥¥f
¥r	¥¥r
‘(シングルクオート)	¥’
“(ダブルクオート)	¥”
¥	¥¥
/	¥/

■攻撃コードで送信される値がエスケープされると・・・・

'%2B@java.lang.Runtime@getRuntime().exec(“notepad.exe")%2B'

↓

¥'%2B@java.lang.Runtime@getRuntime().exec(¥“notepad.exe¥")%2B¥'

⇒OGNL式として解釈されない文字列にエスケープされる。

Copyright©2013 JPCERT/CC All rights reserved.

修正版コード: エスケープ実施後のOgnlUtil::getValueメソッドの動作

StringEscapeUtils::escapeJavaメソッドでエスケープされた値をOgnlUtil::getValueメソッドの引数として渡すことで、OGNL式として評価されてもJavaメソッドは実行されなくなる。

Jspファイル(View) /empmanager/editEmployee.jsp

```
:
<s:textfield label="Employee Id" name="currentEmployee.empId"/>
:
```

OgnlValueStack.java

```
public Object findValue(String expr, Class asType) {
        :
    if ((overrides != null) && overrides.containsKey(expr)) {
            expr = (String) overrides.get(expr);
            :
        Object value = OgnlUtil.getValue(expr, context, root, asType);
            :
    }
```

変数exprはエスケープされているため、OGNL式として評価されてもJavaメソッドは実行されない

¥'+@java.lang.Runtime@getRuntime().exec(¥“notepad.exe¥")+¥'

Copyright©2013 JPCERT/CC All rights reserved.

補足

- 今回は数値型への変換エラーの際の処理であったが、それ以外のエラー処理にも同様の脆弱性が存在する可能性がある。
- Struts2の脆弱性でありながら、その上で動くJAVAアプリケーションの仕様によって影響を受けるかどうかが変わる

Copyright©2013 JPCERT/CC All rights reserved.

まとめ

■ この脆弱性から学べるプログラミングの注意点

- SQLクエリやHTMLなど、異なる処理系にデータを出力する際には、出力先の処理を考慮した無害化が必要
- 無害化を実施しないとSQLインジェクションやクロスサイトスクリプティング等の脆弱性が発生する
- 今回はOGNL式にデータを渡す際に、OGNL式として解釈されないような無害化処理を実施していなかったため、脆弱性が発生した

■ 上記への対策

- データを出力する際には、出力先でデータがどのように解釈されるかを確認する
- 出力先でデータが意図しない解釈をされないように適切な無害化を行う

Copyright©2013 JPCERT/CC All rights reserved.

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
Apache Strus2 における任意の Java メソッド実行の脆弱性
https://www.jpcert.or.jp/securecoding/2012/No.02_Apache_Struts2.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

Copyright©2013 JPCERT/CC All rights reserved.