



A CORSAIRE WHITE PAPER
SECURITY PITFALLS IN STRIPES WEB
APPLICATIONS

Project Reference	Security Pitfalls in Stripes Web Applications
Authors	Stephen De Vries
Date	11 May 2009
Distribution	Client Pre-release

Copyright © 2009 Corsaire Limited. All Rights Reserved.



Security Pitfalls in Stripes Web Applications

Table of Contents

TABLE OF CONTENTS	2
OVERVIEW	3
TECHNICAL INTRODUCTION TO STRIPES.....	4
POTENTIAL SECURITY VULNERABILITIES.....	6
Default Exposure of Classes	6
ActionBean Listing	6
Data Binding	7
Method Calls	8
Calling 'setter' methods.....	9
POSTs can be GETs	10
Conclusions	11
About The Author	12
About Corsaire.....	12



Security Pitfalls in Stripes Web Applications

Overview

The Stripes framework (www.stripesframework.org) is a Java web presentation framework that aims to ease the process of creating Java based web applications, by favouring defaults over verbose configuration and by providing a single backing bean for both properties and methods.

With ease-of-use and configuration through defaults, there is the risk that the security of the application could be compromised. This white paper (or technical note) explores the security weaknesses that developers might inadvertently introduce into an application built with the Stripes framework and also provides a basis for assessing Stripes based web applications from a security point of view.

This paper covers Stripes version 1.5.1 from www.stripesframework.org. It exposes a number of potential security weaknesses that should be included in a comprehensive Web Application Security Assessment, but should not be regarded as a complete methodology for security assessing Stripes based web applications. For a complete methodology consult the OWASP Testing Guide¹.

When security assessing web applications it is strongly recommended that a full-knowledge or white-box approach be taken, where the source code is available to the security testers. This enables testers to identify security issues quickly and accurately and addresses the risk of misidentifying vulnerabilities.

¹ http://www.owasp.org/index.php/Category:OWASP_Testing_Project



Security Pitfalls in Stripes Web Applications

Technical Introduction to Stripes

Stripes uses classes that implement the ActionBean interface to provide remote methods and perform data binding for the web tier. Certain methods in the ActionBean can be called by including them as URL parameters, and the value of the internal variables can similarly be set by including them as URL parameters. All methods that return a *Resolution* type can be called from the web tier. The value of ActionBean properties can be set if they have a corresponding setter method.

For example, consider the following ActionBean fragment:

```
package com.corsaire.example;
public class MessageActionBean implements ActionBean {
    private String msg;
    private String name;

    public Resolution sayHello() {
        msg = "Hello "+name;
        return new ForwardResolution("/index.jsp");
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

To set the *name* variable to “bob” and then call the *sayHello* method, the following URL would be used:

<http://.../StripesExample/com/corsaire/example/message?name=bob&sayHello=>

Note also that if there is no explicit mapping of the URL through the `@UrlBinding` annotation², then Stripes uses the package and class names to construct a default URL that can be used to access the bean. For example, the class: *Calculator* in the package *org.example.beans* would be accessible from the URL: <http://.../org/example/beans/Calculator>. Stripes performs some string manipulation on the class and package names by, for example, removing the trailing strings “Action” and “Bean” from the class name. Other transformations are also applied to the package name. See the *NameBasedActionResolver* documentation³ for more details on the algorithm.

If an ActionBean class uses the `@UrlBinding` annotation, then this default mapping of classes to URLs is disabled and only the specified URL is used.

² <http://www.stripesframework.org/display/stripes/Annotation+Reference#AnnotationReference-@UrlBinding>

³

<http://stripes.sourceforge.net/docs/current/javadoc/net/sourceforge/stripes/controller/NameBasedActionResolver.html>



Security Pitfalls in Stripes Web Applications

The request lifecycle for Stripes is as follows⁴:

1. Resolve an ActionBean based on the URL of the request and set the ActionBeanContext on it.
2. Resolve the Handler method that will handle the event received in the request.
3. Bind properties from the HttpServletRequest into the ActionBean, performing validation as necessary.
4. Invoke any custom validation methods.
5. Invoke the appropriate handler method on the ActionBean.
6. If the ActionBean returns a non-null Resolution, execute it.

⁴ <http://www.stripesframework.org/display/stripes/Lifecycles+Etc>.



Security Pitfalls in Stripes Web Applications

Potential Security Vulnerabilities

Default Exposure of Classes

In the Stripes configuration a package is used to define where all the exposed ActionBean classes are located. All classes within the specified package that implement the ActionBean interface will be exposed by Stripes. This is configured in the web.xml file as follows:

```
<filter>
  <display-name>Stripes Filter</display-name>
  <filter-name>StripesFilter</filter-name>
  <filter-class>net.sourceforge.stripes.controller.StripesFilter</filter-class>
  <init-param>
    <param-name>ActionResolver.Packages</param-name>
    <param-value>my.action.bean.pkg</param-value>
  </init-param>
</filter>
```

This feature could inadvertently allow ActionBeans that were designed for debugging or testing purposes to be exposed in the production application. The packages defined here should be inspected to ensure that only the ActionBeans designed for the production environment are exposed.

ActionBean Listing

Stripes provides a helpful default error message that lists all the exposed ActionBeans. This message is displayed when the StripesDispatcher receives a request for an ActionBean, but where the bean doesn't exist. The URLs that are mapped to the StripesDispatcher are configured in the web.xml file. A typical example would be:

```
<servlet-mapping>
  <servlet-name>StripesDispatcher</servlet-name>
  <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

This maps all URLs that end in ".action" to the StripesDispatcher. Multiple URL patterns can be mapped to the StripesDispatcher servlet too. If the StripesDispatcher servlet cannot resolve a requested URL, such as "<http://.../StripesExample/something.action>" to a suitable ActionBean, then Stripes responds with an HTTP 500 error message similar to the example below:

"net.sourceforge.stripes.exception.ActionBeanNotFoundException: Could not locate an ActionBean that is bound to the URL [/something.action]. Commons reasons for this include mis-matched URLs and forgetting to implement ActionBean in your class. Registered ActionBeans are:

*{/com/corsaire/example/FundsTransfer.action=class com.corsaire.example.FundsTransferActionBean,
/com/corsaire/example/Message.action=class com.corsaire.example.MessageActionBean,
/controller/DefaultView.action=class net.sourceforge.stripes.controller.DefaultViewActionBean}"*



Security Pitfalls in Stripes Web Applications

Listing the available ActionBeans could be useful to an attacker while profiling and planning attacks against the application. This error will only be displayed if error handling for HTTP 500 messages has not been defined in the web.xml file.

Data Binding

In common with other Java web frameworks, Stripes performs data binding directly from submitted form fields or query string parameters to the backing ActionBean as described in the overview above. In addition to binding simple types, Stripes also supports binding complex classes. Consider the example of a complex User type that stores a number of attributes, including an 'admin' Boolean value used to determine whether the user is an administrative user or not:

```
public class User {
    private String firstname;
    private String lastname;
    private boolean admin=false;

    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    public String getLastname() {
        return lastname;
    }
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
    public boolean isAdmin() {
        return admin;
    }
    public void setAdmin(boolean admin) {
        this.admin = admin;
    }
}
```

Typically, this would be used in an ActionBean as follows:

```
public class MessageActionBean implements ActionBean {
    private User user;
    public void setUser(User user) {
        this.user = user;
    }
    public User getUser() {
        return user;
    }

    public Resolution saveUser() {
        userManager.saveUser(getUser());
    }
    ...etc...
}
```

JSP pages can then set the required properties:

```
<stripes:form action="/com/corsaire/example/RegisterUser.action" focus="">
  <stripes:errors/>
  First name <stripes:text name="user.firstname"/><br>
  Last name<stripes:text name="user.lastname"/><br>

  <stripes:submit name="register" value="Register"/>
</stripes:form>
```



Security Pitfalls in Stripes Web Applications

```
</stripes:form>
```

In this case, only the `firstname` and `lastname` properties were included in the form fields, but since the entire `User` class can be set on the `ActionBean`, it is possible to set any of its properties directly, such as:

```
http://.../RegisterUser.action?register=&user.firstname=bob&user.lastname=smith&user.admin=true
```

This form of nested binding can be restricted and controlled from within Stripes by using the `@StrictBinding` annotation, but this is not enabled by default. See <http://greggbolinger.blogspot.com/2008/01/stripes-15-feature-control-binding-with.html> for more details.

Method Calls

Web applications are often designed with functions that expect a certain order in the methods that are called and the variables that are set. The order of such a workflow is often built into the presentation layer of the application by limiting the links and forms that a user sees on the page. From a security perspective, it is useful to discard this intended workflow and instead view a Stripes web application as a form of remote method invocation (RMI) where variables can be set and methods called in any order.

Consider the following fragment of an `ActionBean` from a (slightly contrived) banking application that first requires a funds transfer to be approved before initiating it:

```
public void setFromAccount(String fromAccount) {
    this.fromAccount = fromAccount;
}

public void setToAccount(String toAccount) {
    this.toAccount = toAccount;
}

public void setAmount(BigDecimal amount) {
    this.amount = amount;
}

public Resolution initiateTransfer() {
    if (accountService.isApproved(transferId)) {
        accountService.initiateTransfer(fromAccount, toAccount, amount);
    }
    return new ForwardResolution("/transfer.jsp");
}

public Resolution approveTransfer() {
    accountService.approveTransfer(fromAccount, toAccount, amount);
    return new ForwardResolution("/transfer.jsp");
}
```

The required workflow could be implemented by first presenting the user with a form that results in the following request:

```
http://.../FundsTransfer.action?approveTransfer=&fromAccount=123&toAccount=456&amount=10
```



Security Pitfalls in Stripes Web Applications

When the form is submitted, the `fromAccount`, `toAccount` and `amount` values are stored in the HTTP session. Then on a subsequent page, another form for the following request is provided:

<http://...../FundsTransfer.action?initiateTransfer=>

The flaw with this approach is that setting the bean properties and the order of the workflow is effectively enforced on the client-side, by simply presenting the forms in a specific order. And since both the `initiateTransfer` and the `approveTransfer` methods are in the same ActionBean, Stripes will re-bind any query string parameters for either of the methods. This workflow could therefore be subverted by an attacker by simply calling the `initiateTransfer` method directly and setting arbitrary values for the `fromAccount`, `toAccount` and `amount` properties thereby bypassing the approval method:

<http://...../FundsTransfer.action?initiateTransfer=&fromAccount=123&toAccount=456&amount=10000>

Attackers can call any methods on an ActionBean that return a 'Resolution' type and set any properties that have a corresponding setter method, in any order.

Calling 'setter' methods

Java bean 'setter' methods are used to set properties on ActionBeans as illustrated in the example above with the `setAmount`, `setToAccount` and `setFromAccount` methods being used to set the `amount`, `toAccount` and `fromAccount` properties respectively. The JavaBean conventions imply that there is an internal property X that is changed when calling a `set<X>` method.

Stripes, however, is quite liberal in how it matches URL query parameters to ActionBean setter methods, specifically any method that starts with the letters 'set' and is declared as 'public' and 'void' is regarded as a setter method.

To illustrate why this could be a security concern, consider the following fragments from an ActionBean:

```
public void settings (String mySettings) {
    settingsManager.update(mySettings);
}
```

The method is probably not intended to be a JavaBean setter method, but because it fulfils Stripes' requirements by starting with the letters 'set' and has a 'public void' return type, it can be called from a URL:

<http://...../SomeBean.action?someMethod=&tings=hello>

By apparently trying to set the 'tings' property, the 'settings' method is called with the supplied String parameter 'hello'.



Security Pitfalls in Stripes Web Applications

This could allow attackers to potentially access methods that were not intended to be exposed.

POSTs can be GETs

By default, HTTP POST and GET methods are interchangeable in a Stripes application because Stripes merges URL parameters with parameters from the body of the request. If both parameters are provided, then those in the URL override those in the request body during data binding, for example, the request:

```
POST
http://localhost:8080/StripesExample/com/corsaire/example/RegisterUser.action?user.firstname=bob
HTTP/1.1
...
Content-Type: application/x-www-form-urlencoded
Content-Length: 115
user.firstname=alice&user.lastname=d&register=Register&_sourcePage=gTVKqsr4AMa-
SN1GgtNqAoNOTmcafST2&__fp=THhINzPB1gk%3D
```

The user bean will be assigned the firstname of bob, not alice.

This feature could be useful to attackers when exploiting Cross Site Scripting⁵ or Cross Site Request Forgery⁶ vulnerabilities in the application. Currently, this behaviour is built into Stripes and would require modification of the source code in order to change it.

⁵ http://www.owasp.org/index.php/Cross_site_scripting

⁶ http://www.owasp.org/index.php/Cross-Site_Request_Forgery



Security Pitfalls in Stripes Web Applications

Conclusions

There are a number of potential security pitfalls in the Stripes framework that could allow an attacker to subvert the intended application functionality, directly edit sensitive data or access functionality that should not have been exposed. These issues can be avoided by understanding the security implications of some of Stripes' features. These potential issues can also be identified and addressed during a security code review or application security assessment.



Security Pitfalls in Stripes Web Applications

About The Author

Stephen de Vries is a Principal Consultant in Corsaire's Security Assessment team. He has worked in IT Security since 1998, and has been programming in a commercial environment since 1997. He has spent the last eight years focused on Security Assessment and Audit at Corsaire, KPMG and ISS. He was also a contributing author and trainer on the ISS Ethical Hacking course. He is currently leading the OWASP Java Project and regularly presents on secure programming and testing.

Stephen's past roles have included that of a Security Consultant at a leading City of London Financial institution and also Security Engineer at SMC Electronic Commerce. At both positions he was involved in corporate security at many levels and was responsible for consulting on the paper security policies and procedures, conducting vulnerability assessments, designing, deploying and managing the security infrastructure of the organisation.

About Corsaire

Corsaire are experts at securing information systems, consultancy and assessment. Through our commitment to excellence we provide a range services to help organisations protect their information assets and reduce corporate risk. Founded privately in the United Kingdom in 1997, we operate on an international basis with a presence across Europe, Africa and the Asia-Pacific rim. Our clients are diverse, ranging from government security agencies and large blue-chip FTSE, DAX, Fortune 500 profile organisations to smaller internet start-ups. Most have been drawn from banking, finance, telecommunications, insurance, legal, IT and retail sectors. They are experienced buyers, operating at the highest end of security and understand the differences between the ranges of suppliers in the current market place. For more information contact us at contact-us@corsaire.com or visit our website at <http://www.corsaire.com>.