Security

THIS PAGE IS DEPRECATED, PLEASE FOLLOW THE LINK TO THE NEW SECURITY GUIDE!

http://struts.apache.org/security/

Security tips

The Apache Struts 2 doesn't provide any security mechanism - it is just a pure web framework. Below are few tips you should consider during application development with the Apache Struts 2.

Restrict access to the Config Browser

Config Browser Plugin exposes internal configuration and should be used only during development phase. If you must use it on production site, we strictly recommend restricting access to it - you can use Basic Authentication or any other security mechanism (e.g. Apache Shiro)

Don't mix different access levels in the same namespace

Very often access to different resources is controlled based on URL patterns, see snippet below. Because of that you cannot mix actions with different security levels in the same namespace. Always group actions in one namespace by security level.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>admin</web-resource-name>
        <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Never expose JSP files directly

You must always hide JSP file behind an action, you cannot allow for direct access to the JSP files as this can leads to unpredictable security vulnerabilities. You can achieve this by putting all your JSP files under the `WEB-INF` folder - most of the JEE containers restrict access to files placed under the `WEB-INF` folder. Second option is to add security constraint to the `web.xml` file:

```
<!-- Restricts access to pure JSP files - access available only via Struts action -->
<security-constraint>
    <display-name>No direct JSP access</display-name>
    <web-resource-collection>
        <web-resource-name>No-JSP</web-resource-name>
        <url-pattern>*.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>no-users</role-name>
    </auth-constraint>
</security-constraint>

<security-role>
    <description>Don't assign users to this role</description>
    <role-name>no-users</role-name>
</security-role>
```

The best approach is to used the both solutions.

Disable devMode

The `devMode` is a very useful option during development time, allowing for deep introspection and debugging into you app.

However, in production it exposes your application to be presenting too many informations on application's internals or to evaluating risky parameter expressions. Please **always disable `devMode`** before deploying your application to a production environment. While it is disabled by default, your `struts.xml` might include a line setting it to `true`. The best way is to ensure the following setting is applied to our `struts.xml` for production deployment:

> ⚠ **How to disable devMode in production**
>
> <constant name="struts.devMode" value="false"/>

Reduce logging level

It's a good practice to reduce logging level from **DEBUG** to **INFO** or less. Framework's classes can produce a lot of logging entries which will pollute the log file. You can even set logging level to **WARN** for classes that belongs to the framework, see example Log4j2 configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="STDOUT" target="SYSTEM_OUT">
            <PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="com.opensymphony.xwork2" level="warn"/>
        <Logger name="org.apache.struts2" level="warn"/>
        <Root level="info">
            <AppenderRef ref="STDOUT"/>
        </Root>
    </Loggers>
</Configuration>
```

Use UTF-8 encoding

Always use `UTF-8` encoding when building an application with the Apache Struts 2, when using JSPs please add the following header to each JSP file

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

Do not define setters when not needed

You should carefully design your actions without exposing anything via setters and getters, thus can leads to potential security vulnerabilities. Any action's setter can be used to set incoming untrusted user's value which can contain suspicious expression. Some Struts `Results` automatically populate params based on values in `ValueStack` (action in most cases is the root) which means incoming value will be evaluated as an expression during this process.

Do not use incoming values as an input for localisation logic

All `TextProvider`'s `getText(...)` methods (e.g in `ActionSupport`) perform evaluation of parameters included in a message to properly localize the text. This means using incoming request parameters with `getText(...)` methods is potentially dangerous and should be avoided. See example below, assuming that an action implements getter and setter for property `message`, the below code allows inject an OGNL expression:

```
public String execute() throws Exception {
    setMessage(getText(getMessage()));
    return SUCCESS;
}
```

Never use value of incoming request parameter as part of your localisation logic.

Internal security mechanism

The Apache Struts 2 contains internal security manager which blocks access to particular classes and Java packages - it's a OGNL-wide mechanism which means it affects any aspect of the framework ie. incoming parameters, expressions used in JSPs, etc.

There are three options that can be used to configure excluded packages and classes:

- `struts.excludedClasses` - comma-separated list of excluded classes
- `struts.excludedPackageNamePatterns` - patterns used to exclude packages based on RegEx - this option is slower than simple string comparison but it's more flexible
- `struts.excludedPackageNames` - comma-separated list of excluded packages, it is used with simple string comparison via `startWith` and `equals`

The defaults are as follow:

```
<constant name="struts.excludedClasses"
          value="com.opensymphony.xwork2.ActionContext" />

<!-- this must be valid regex, each '.' in package name must be escaped! -->
<!-- it's more flexible but slower than simple string comparison -->
<!-- constant name="struts.excludedPackageNamePatterns" value="^java\.lang\..*,^ognl.*,^(?!javax\.servlet\..+)
(javax\..+)" / -->

<!-- this is simpler version of the above used with string comparison -->
<constant name="struts.excludedPackageNames" value="java.lang,ognl,javax" />
```

Any expression or target which evaluates to one of these will be blocked and you see a WARN in logs:

```
[WARNING] Target class [class example.MyBean] or declaring class of member type [public example.MyBean()] are
excluded!
```

In that case `new MyBean()` was used to create a new instance of class (inside JSP) - it's blocked because `target` of such expression is evaluated to `java.lang.Class`

It is possible to redefine the above constants in `struts.xml` but try to avoid this and rather change design of your application!

Accessing static methods

Support for accessing static methods from expression will be disabled soon, please consider re-factoring your application to avoid further problems! Please check WW-4348.

OGNL is used to call action's methods

This can impact actions which have large inheritance hierarchy and use the same method's name throughout the hierarchy, this was reported as an issue WW-4405. See the example below:

```
public class RealAction extends BaseAction {
    @Action("save")
    public String save() throws Exception {
        super.save();
        return SUCCESS;
    }
}

public class BaseAction extends AbstractAction {
    public String save() throws Exception {
        save(Double.MAX_VALUE);
        return SUCCESS;
    }
}

public abstract class AbstractAction extends ActionSupport {
    protected void save(Double val) {
        // some logic
    }
}
```

In such case OGNL cannot properly map which method to call when request is coming. This is do the OGNL limitation. To solve the problem don't use the same method's names through the hierarchy, you can simply change the action's method from `save()` to `saveAction()` and leaving annotation as is to allow call this action via `/save.action` request.

Accepted / Excluded patterns

As from version 2.3.20 the framework provides two new interfaces which are used to accept / exclude param names and values - AcceptedPatternsChecker and ExcludedPatternsChecker with default implementations. These two interfaces are used by Parameters Interceptor and Cookie Interceptor to check if param can be accepted or must be excluded. If you were using `excludeParams` previously please compare patterns used by you with these provided by the framework in default implementation.

Strict Method Invocation

This mechanism was introduced in version 2.5. It allows control what methods can be accessed with the bang "!" operator via Dynamic Method Invocation. Please read more in Strict Method Invocation section of Action Configuration.