HACKING JAVA-WEBAPPS FOR DUMMKOPFS

# Hacking Java-Webapps for Dummkopfs

While PHP dominates the web development ecosystem, many competitors such as NodeJS, Ruby and Python have risen against it. One of the oldest competitors is Java, an enterprise grade Object-Oriented Programming (OOP) language, which can be run in many environments. Because of its platform-independent model and enterprise grade programming, it is used in many corporations and state-driven projects as the language of choice.

If you want to perform a run on Arasaka Inc. or any other megacorp, you need to know how to hack Java webapps.

## Java Webapps for Non-Java People

Much like JAR files, Java webapps are contained in WAR files which contain all of the Java object code, static images, css, JSP pages and servlets. These WAR files can then be deployed on Java application servers, such as the popular Apache Tomcat or WildFly (previously known as JBoss). The application server handles the bootstrapping and parsing requests and forwards them to the code contained in the WAR archive.

### What's inside a WAR?

In a nutshell, a WAR archive can be free form, and there isn't a one standard place for different files. However, things that are usually in the WAR file are the directories WEB-INF and META-INF. These contain configuration files the application server, such as Tomcat, uses to parse the archive and map the handlers.

Example of a simple WAR file:

```
.war/
WEB-INF/
... web.xml
... struts-config.xml
... classes/
... lib/
META-INF/
... context.xml
```

The WEB-INF can also contain precompiled classes (.class files) in a directory named "classes" and third-party libraries in the directory called "lib". The most central file in the WEB-INF directory is the file called "web.xml". This configuration file contains basic information about the different settings, filters and servlets contained in the archive.

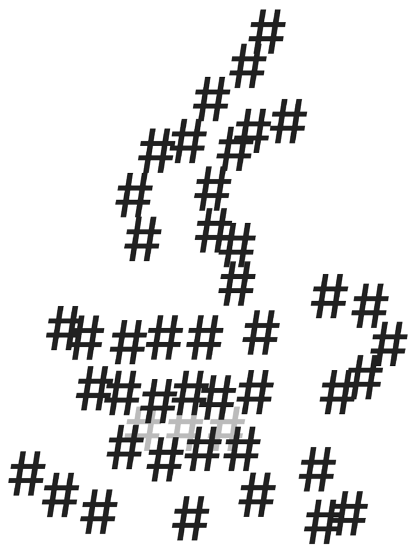
The other important configuration file in WEB-INF is the struts-config.xml file which contains mappings of different requests (for example /hello) to different handlers (or servlets, such as com.company.application.class). It also has configuration to different login handlers, redirects and JavaBeans (more on that later).

The META-INF directory may or may not exist in the WAR file. This directory usually contains a file called "context.xml" that is used to configure application wide settings, such as database connections, which can be used by any servlet as a data source. This information can also reside in the application servers config directory, making the configuration options server-wide and not just application-wide.

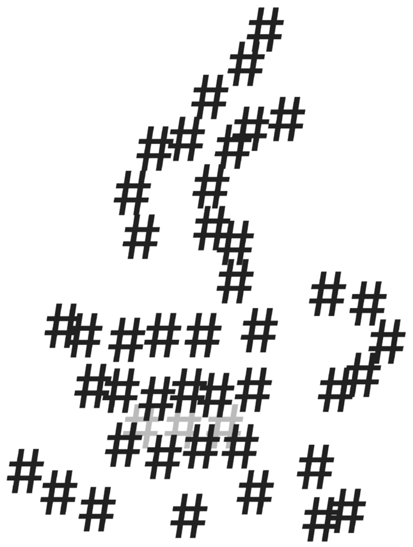
Static data, such as images and css, can pretty much exist anywhere in the archive.

### Servlets, JSPs? Beans?
Webapps written in Java usually try to emulate an object-oriented approach to web development. While there are .jsp files, which can be thought of like your basic .php files, mixing

HACKING
JAVA-WEBAPPS
FOR DUMMKOPFS



HACKING
JAVA-WEBAPPS
FOR DUMMKOPFS



normal HTML with dynamic Java code, the salt of the application is servlets. Servlets are like normal Java classes, except that they take in HTTP requests and spit out HTTP responses. Usually they go even deeper and try to distinguish servlets which output HTML pages with the servlets that perform actions, such as database queries, with the virtual file extension of ".do". You can think of the .do files as getters and setters, if you know your Object-Oriented Programming.

Example mapping of servlets:

```
index -&gt; com.company.app.showIndex
showNews.do -&gt; com.company.app.getNews
postNews.do -&gt; com.company.app.postNews
```

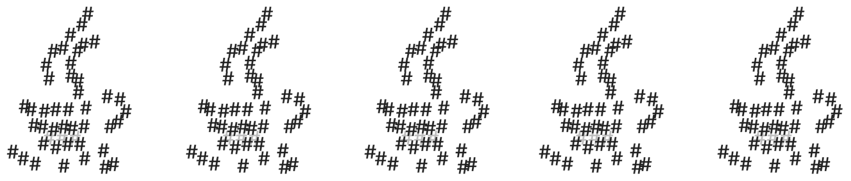
While the JSP pages are like php, mixing HTML with the actual serverside code, servlets usually use a library called JavaBeans. JavaBeans is a simple way to render and construct HTML code serverside, by telling it what you need, whether it is an HTML form or a static image.

## So how do we hack it?

### Application server

A lot of information already exists of this, but I'll tell you the answer:

You need to brute-force the login, then upload



your malicious backdoor WAR application. There is no silver bullet for this and you need to refer to the documentation of your application server. For example, the Tomcat admin runs usually on port 8080, where you can try user/pass combinations such as:

```
admin:admin
admin:tomcat
tomcat:tomcat
```

And so on an so forth. Metasploit has modules for all of this.

### Servlets
The most common (and destructive) Java webapp bugs in order are:
1. Access control
2. Local file/resource disclosure
3. SQL injection

SQL injection is a bug that I won't be talking about in this article, as it is a common flaw and can be exploited in the same way as in PHP.

### Access control, or how I logged in as an admin
While Java has its ways (JSESSID) to control logging in an out of an system, it's up to the developer to keep track of which parts of the webapp an user can and cannot enter. This can be either done in the servlets or as a filter which is applied in web.xml/struts-config.xml. Either way, you can never be too sure that the

developer didn’t leave something out.
For example, if we see that the page:
/adminPanel
Is password protected, we should check and see if pages associated with it are. If you have the config files (more on that later), you can go and check every page for access control vulnerabilities. If not, you can generate a huge word list of blind tests, such as:

```
adminDashboard
adminDashboard.jsp
adminDashboard.do
userAdd
userAdd.jsp
userAdd.do
```

And so on and so forth, and check if you can find anything interesting. One of the common mistakes developers make in this object-oriented ecosystem is having access control in your basic servlets, but not your getters/setters (the .do mappings). If not, you can also test the found servlets for other vulnerabilities.

### Local file/resource disclosure

This is a classic mistake, but in Java, it’s so easy to make, especially on the resource side. If you see servlets like this:

```
showPage.do?page=asd.jsp

image.jsp?image=123.png

userRegister?step=sendEmail.do
```

You are bound to find one of these vulnerabilities. The difference between resource and file disclosure is that in a file disclosure, the code is using a filestream to open the file, meaning you can read any file there is, such as /etc/passwd. In a resource disclosure, the servlet is opening a resource inside of the WAR archive, and you are limited to browsing inside the archive.

Try to use backhops like "../" and see what you find. Fimap may also work.

#### File Disclosure
File disclosure is bad, real bad, if you know how to use it. In order to exploit it, you need to know the operating system/distribution the Java application is running on, because even in linux distributions, the system files may be kept in different directories.

Try to:
1. Get system information: hostname, network configuration, anything in /proc/, bashrc...
2. Configuration files, sshd, apache, ftpd...
3. Password/gpg files, can be found using the configuration files
4. Log files, everything in /var/log/

You can find a lot of juicy information which will surely help you break into the megacorporation.

#### Resource disclosure
This one is a bit trickier, as you are limited to working in the current WAR file. Nevertheless, you can still find juicy information.

Try to:
1. Get the configuration files WEB-INF/web.xml, WEB-INF/struts-config.xml
2. Possible database passwords in META-INF/context.xml
3. Static files, if you can find them

These files contain information on how the webapp works from within and can contain useful information, such as FTP/database information, user accounts, test/debug servlets, logs and the like.

More information

Apache Tomcat documentation:

https://tomcat.apache.org/tomcat-7.0-doc/

OWASP: https://www.owasp.org/index.php/

Category:Java

Metasploit: https://www.metasploit.com/

Fimap: http://www.fimap.com/

Google: https://startpage.com/