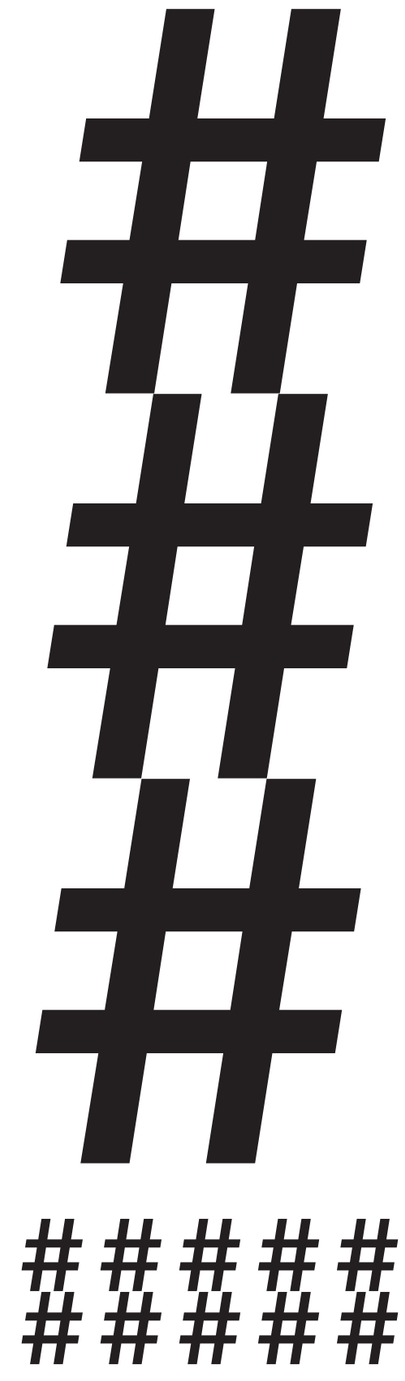
lainzine.org

LAINZINE

VOL FIVE

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.

go-outside.txt

Slowly, one by one, we used this to liberate any like minds we met. No longer would you have to find a printing press to post your propaganda; subversive ideas and forbidden connections were now in the bedroom and the palms of our hands. Although the pleasures of our basic desires were distracting, the ecstasy of our higher ones drove us to the furthest reaches of cyberspace in search of friends, comrades, lovers. As more and more of the physical world connected, the power of those minds Wired together grew, and we reached back into the ruins of our past to brighten those darkened hideaways and defy the so-called "reality" that had been imposed on us. These new found interfaces gave us the knowledge and the resources to do things like earn a wage without paying our dues to the social convention, hack our own neural networks with designer substances, affect the physical world in ways never seen, and for the first time in our lives - or anyone's for that matter - shape society's dialogue with our keystrokes.

The voice we synthesized for ourselves was loud, clear, and threatening; so threatening, in fact, that those oppressors we thought we had escaped feared we could not be beaten and joined us. The moneyed monoliths brought with them soon dragged us into the knowing nightmares of our earlier lives. What lucky few were chosen to be society's new upper echelon by the insular elite were sold for the promise of safety, comfort, the security of our future - and a few other lies. I wonder if we flocked to this simply because we knew fleeting pleasures and our greatest fears more than we knew what to do with ourselves once we were finally able to be alone with eachother, whether we warped our heady ideals into their antithesis or if we simply lost hope. In any case, it is certain that this space between the fiber-optics and spinning platters is no longer ours either. It was taken just like our living rooms, leaving another unfillable space in our cramped highrise apartments.

Some of us still hide, whispering in the new dark corners of what we have built. We ruminate about what we didn't know that hurt us, how to start over and create a better world where "reality" would be something in which all those children we aren't or shouldn't be having will revel and explore. We tangle and bond with the mess of wires until they cut us, hoping someone as trapped as we are will taste freedom in what comes out, but most of those dreaming kids are still scattered and alone, unable to bridge our homes in the Wired world with the sensory one. Every once in a while, a few of us find a corner without being followed by those masses who tell us not to touch the rat's nest of connections lest we sever one of the countless, long-dead strands slicing into our ability to live, in the wishful belief that there are still a few thinking people somewhere out there, and send it back in hopes that others will join us in the same way that *we* were liberated.

But no-one answers anymore. Cyberpunk is dead. If you don't believe me, see it for yourself.

Just go outside.

go-outside.txt

By: Hisui Kohaku

I don't know if this was really an experience of yours as a kid, but my friends and I were told to "turn off the screen and go outside," as if we weren't socializing enough indoors or something. Begrudgingly, we'd leave our games or anime on pause and go out until we were let back into our fantasy realms. We did socialize, but much of the time, it was just about that: our virtual worlds; the ones we were *really* living in, where we achieved great triumphs and people actually cared about our lonely tragedies. We found a short, brutal middle ground between our childhood's "I want to be an astronaut!"and our adulthood's "I want to be out of debt" that we held for dear life as "reality " crumbled around us, and it was all was in front of a cathode ray tube.

But we had a strange (youthful, flawed) way of systems thinking about these two realities. It's not that we had zero interest in the outside; when we were kicked out into the undesigned physical realm, beyond the supervision of our overtired parents, we did make some agency for ourselves with graffiti, fistfights, and bummed cigarettes. That ground we fought so hard to defend had been lost to unwanted younger brothers, parents claiming their primetime shows, drunken shouting in the kitchen, too much homework. Those idealistic children who were told they could be anything had chosen to be destitute second-rate punks flung across suburbs and dormitories over becoming tomorrow's struggling middle-managers of mediocrity; that is, they would rather suffer unwatched than endure the truthful but ugly version of the surveilled future they had been promised when their biggest worries involved waking up early enough for Saturday morning cartoons.

And then one day, a childhood dream came from the past to wake us up. Those kids who saw a generator in Home Depot and ever since yearned to take the game beyond the living room and weave it into the emptiness of physical life, the ones who wished they had their own, *private* screen with which to build any edifice they liked, finally got an answer besides an adult platitude or a dial tone. Devices small and cheap enough to be handed down for the sake of keeping up with Joneses or purchased with scrounged cash were widely available and the future of business forced our parents to let us have them. Our communications were private so long as we fled to the next platform in the never-ending line of chatrooms, messengers, and message boards that kept us above people deciding who we could and couldn't talk to. The quietly renegade attitudes that had us loitering in the forgotten corners of our parent's greatest creations led us to make our own, and our increasing skills of secrecy let us create it in the image of the secret selves revealed when the devices became a part of us.

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).

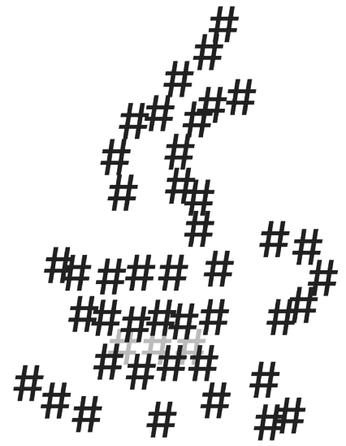
The META-INF directory may or may not exist in the WAR file. This directory usually contains a file called “context.xml” that is used to configure application wide settings, such as database connections, which can be used by any servlet as a data source. This information can also reside in the application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

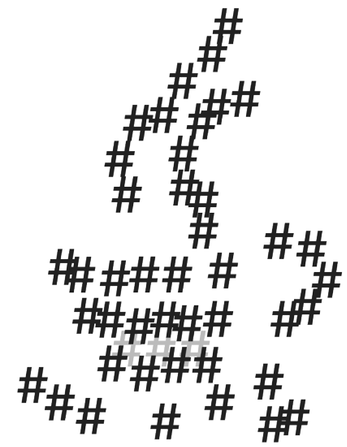
Servlets, JSPs? Beans?

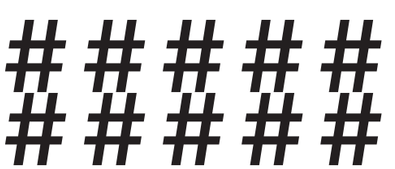
Webapps written in Java usually try to emulate an object-oriented approach to web development. While there are .jsp files, which can be thought of like your basic .php files, mixing

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


```
# 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/
```

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.

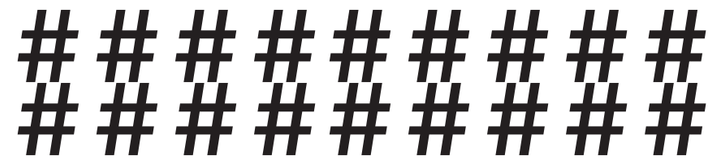
Servlets

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.