Java JAR Attacks and Features

Java JAR Attacks and Features - pdp, gnucitizen.org.

- Published: date not stated
- Original: <https://www.gnucitizen.org/blog/java-jar-attacks-and-features>
- Current location: <https://www.gnucitizen.org/blog/java-jar-attacks-and-features/>
- Preserved from: https://www.gnucitizen.org/blog/java-jar-attacks-and-features/ (stored) on 2026-08-11
- 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.

Java JAR Attacks and Features

Sat, 10 Nov 2007 22:35:33 GMT

by pdp

This is my last post for this week, so I will try to keep it short and sweet. While playing with the JAR protocol for Firefox (here and here), I also did some investigation on the way Java handles jars, as well. To my surprise, the Java runtime seems to posses some very interesting features which may easily be implemented into an attack scenario. I shared some of my research and ideas with Aviv R aff and we both concluded that the observed behavior may not be the perfect way to attack someone, although IMHO, I see nothing more but opportunities for **evilness**.

Here is the deal. If someone manages to upload an Applet, as a JAR archive, on the targeted server, they will be able to invoke it from a page located on a different server/site. "This is not very exciting but we are getting to the interesting part soon."

Many Web servers come with quite a few open ports which are there for management purposes only (backup consoles, MsSQL or any other backend database, SMB, etc). Unfortunately, these services are behind the firewall, i.e. we can see only ports 80/TCP and maybe 443/TCP but nothing else, the rest is simply blocked for external IP address. However, if you are behind the firewall, well, no such restrictions are applied. This means that internal visitors have more access to the remote server then external visitors.

When a user invokes an applet, this applet will run within the sandbox of the `codebase` parameter, i.e the applet can only access resources that are located on the IP address/domain it was delivered from. This also applies to socket communication. For example, if we open an applet from

`http://example.com:80`, the byte code will be able to connect with a socket to `example.com:25` without the need for extra permissions. This is how the Java's same origin policies are implemented. There is a lot more than this but you get the point.

Having this in mind, it is obvious that attackers can abuse Java to poke a server as being accessed behind the firewall. The attack of course is very simple. The attacker needs to upload the JAR blob or a `.class` file on the targeted server. Then the attacker needs to host a page that embeds the applet somewhere (does not have to be the targeted server) and get someone from behind the firewall to visit it. When the victim/proxy opens the page, the attacker will be able to poke the targeted server from behind the firewall. So in case the targeted server runs MsSQL with a blank `SA` password, the attacker will be able not only to compromise the backend database but also execute commands on the backend itself and as such get some sort of remote command execution. Of course, this all depends on how the MsSQL instance is setup, but most of the time, these services are very poorly configured, so attackers have some real potentials with this technique.

Of course, this is only one of the things the attackers can do to the target. Because applets have different origin checks from JavaScript and Web pages in general, attackers can bridge two zones together (the origin of the applet and the origin of the hosted malicious page). Therefore, it is possible to create some kind of proxy which will allow the attacker to access any service on the target, again as seen by the victim.

Aviv, pointed out that there are two many IFs, and I must agree with him. The first IF is related to whether the target can host user supplied files. This might be the case, but there also will be some kind of restriction on what files the users can upload (i.e. jpg, gif, doc, etc, etc). The second IF is related to how to get someone behind the firewall to visit the evil page. There are ways, which often involve social engineering and a bit of strategy, but I will leave this blank space for you to fill it yourself. Finally, the 3rd IF is related to whether there is anything interesting on the target. So, it kind of looks like a trick which is really hard to pull. But, I kind of like these the most.

IF #1 - uploading the malicious JAR/Applet

If the target does not have any file upload facilities then the attacker is left with nothing but to switch to another strategy. Keep in mind that this attack is tailer made for sites that allow uploads. The second obstacle is to trick the remote upload facility that you are uploading the right type of file while you are delivering something completely different at the end. This is where the second oddity, which I've discovered inside the Java runtime, comes into place.

Simply put, the Java runtime recognizes images as JARs, while the browser or any other software will recognize them as images. "How is that possible?" To answer that, we have to look at the following example. Follow the steps as described bellow:

- Get an image from the Web:

```
fancyimage.jpg
```

- Prepare your jar:jar cvf evil.jar Evil*.class

- Put them together: copy /B fancyimage.jpg + evil.jar fancyevilimage.jpg or cp fancyimage.jpg fancyevilimage.jpg cat evi.jar >> fancyevilimage.jpg

If you double click on the `fancyevilimage.jpg` you get your default image viewer with the actual image displayed inside. If you put the image inside the `src` attribute of an `img` tag, surprise, surprise, it renders. If you try to verify the image headers, you will get `OK`. However, if you change the extension from `.jpg` to `.zip` and try to unzip it with WinRAR or the command line unzip utility, you get the archive content. So, it seams that uploading weired stuff on a server does not have to be as hard as it seams. In our case, the Java runtime, will happily interpret an image as a JAR.

It is actually possible to attach any other file before the JAR but you have to be careful since you may end up with corrupted JAR. If the remote site does not have any type checks, then the attacker can simply upload a .class file. It works pretty much in the same way.

As I side note, the same technique can be used to trick application into running arbitrary Java classes.

IF #2 - tricking the user into visiting the malicious resource

Come up with something. I really enjoyed the social science classes at my University. Keep in mind that social engineering is not only about telling lies on the phone or via an e-mail. There is a lot more into that like behavioral pattern analysis, for example.

IF #3 - dancing with the devil

The attacker might end up with nothing pretty much. This is the reason why the applet needs to be as sharp as it can get. The applet needs to scan ports, identify services, brute force passwords and pretty much whatever else is needed. The attacker may have only one shot so they will probably throw the kitchen sink at the server.

This is it. I know that it sounds too complicated to be feasible but, hey, if it is that complicated no one will think about it, so the chances of remaining undetected are very high. Also the JAR manipulation technique can be used in a number of other cases to achieve more devastating effect. BTW, if you are around San Jose next week, come for a chat. I will be there pretty much the whole next week at the USA OWASP conn.
