

# [Creating a java bot](https://assignbuster.com/creating-a-java-bot/)

Seeing as jagex finally got off their fat buts and decided to addin some deob and action forcing detection we can’t use them methods any more. People have tried to figure out ways of bypassing this detection, but it’s pretty much impossible. You see, jagex managed to add an event logger into their client, this allows them to see mouse clicks and such; this means that if you suddenly attack an NPC and you haven’t clicked it, they know your autoing. Similarly, for deobed clients, they can detect if method modifiers have changed, i. e.

when “ final static void kaboom(int i)” is altered to “ public static void kaboom(int i)” they know it’s been changed and give you a banstick. Bytecode bots, unfortunately just as bigger failure, if the fact that they take ages to update doesn’t put you off, how about the fact they can detect if you run from main() and if you use the loader / navigate through the RS website (note: this also effects deobs). So well, where does that leave us? Fugly color clickers? Nah, fortunately for us, we can create a bot that doesn’t use deobs, uses the RS website and loader, and injects its bytecode at runtime. “ WOWOHWOW” yes . Any bytecode library should do the job, but in this tutorial I will be using “ BCEL”, seeing as it’s fairly efficient, and pretty flexible (with a good API doc). Part B: How the bot worksOkay, now you may be thinking how do we do this then, well heres the theory: First off, we navigate through the RS site, using a “ Spoof Browser”, to look like a real user using a real browser (in this tutorial i’ll be spoofing firefox, but it’s up to you).

The bot will then select the world that you would like (by parsing the html code and finding the URL (atleast this tutorial/my bot does, ARGA finds the url from a world list)). The bot will then load the RS loader (the archive and code files will also be parsed from HTML). The loader will then proceed to load in the RS classes, we then, right under the loaders noise (infact this is done slightly before the process is started), inject some nifty code that makes the loader load the classes through “ our class loader” rather than java. lang. ClassLoader. This ultimately allows us to “ swap” in our preloaded class files (these will have undergone the transformations (before the whole sequence starts, this can be done through a one off standalone updater or as my bot, and this tutorial will, before the applet is loaded (thus every time the bot is loaded).

Now we have our bot running, “ yay” this means that our AccessorMethods (and scripts that extend it) can access all the bots functions such as sendPrivateMessage(String s) etc. etc. via instance reference and a nifty little interface. To summarise: Bot Loads through batch -; Preloads a “ static RS jar” -; navigates through RS parsing the HTML for required info -; loads RS -; swaps in our “ preloaded classes” -; accesses them through instance reference (using an interface)There are also a couple of other things, rather than action forcing, we “ click”, almost like a color clicker but not quite. We still use similar methods to the days of action forcing, such as getting the objects co-ords, we then do a “ World to Screen” conversion, and click around the result (using some human like algorythm). In this tutorial I will send the events down the event queue for ease, however there are several methods.

This method also means that if programmed well enough, the bot is also capable of updating itself. However obfuscation makes this a PAIN IN THE ASS. Section 2 – Building the basic appletPart A: Building the basic appletOk, I thought you might be eager to get into some programming, so I thought we’d get started, nothing to deep mind; we have more theory to learn before thatWe are going to build the simple applet loader, this will be detectable because we won’t introduce the loader hack until section 3. I presume you know java to a suitable level, can script fluently for ARGA and have made several programs before (old bots help). I also recommend using a suitable folder structure and a good IDE, I will use IDEAJ.

Okay, so get yourself something like this setup: Code: package com. fusionbot. bot;/\*\*@FusionBot by ownagesbotThis code is not allowed to be used without explicit permission by the ownerThe code is wrote purely for educational purposesAll usage must fully credit the owner\*/public class Bot {int world; public static void main(String[] args) {if(args. length == 1)new Bot(Integer. parseInt(args[0])); elseSystem.

out. println(“ Usage: Bot ”);}public Bot(int world) {this. world = world;}}Just a bit of basic OOP to begin with. Okay, so every applet needs an AppletStub, this tells it the parameters from whatevers calling it, the codebases and all the information the applet needs, so it’s time to build one. UnderCode: this.

world = world; Add an instance reference to your bot applet stub, appropriate naming should be used, for such I recommend BAS or something. With the class name BotAppletStub. For now, BotAppletStub should look something like this: Code: package com. fusionbot.

bot;/\*\*\* @FusionBot by ownagesbot\* BotAppletStub. java\* This code is not allowed to be used without explicit permission by the owner\* The code is wrote purely for educational purposes\* All usage must fully credit the owner\*/public class BotAppletStub {public BotAppletStub() {}}Now as you know, an Applet uses an AppletStub, not just some fuckup class we’ve made, so we must fulfill the criteria, so import java. applet.\*; and implement AppletStub, if you usea good IDE it’ll create the methods for you, and all you do is sort them out. Now, I don’t want to do this bot for you, but i’ll give you some tips: getCodeBase() will be your world url (including http:// and the trailing /)getDocumentBase() will be something similar but with “/lang/en/aff/runescape/game. ws? lowmem= 1&plugin= 0” added to the end (depends which way you come from, dont get this mixed up or you risk detectability)Some parameters, infact most don’t need anything filling in, just returning a false boolean should sort some (or as I look at my code, 1 of them).

However, the most important one(s) are setParameter(String, String), and getParameter(String, String), at first this required some slight thinking but I soon figured it out, simply make a hashmap and your sorted, heres a spoiler: HashMap; String, String; parameters = new HashMap; String, String;(); public void setParameter(String key, String value) { parameters. put(key, value); }public String getParameter(String key) {if(parameters. get(key) != null)return parameters. get(key); return “”;}Appologies if the indenting fucks up. The next thing to do is set your parameters using your BAS reference, i’ll give you the first 3.

Code: BAS. setParameter(“ width”, “ 765”); BAS. setParameter(“ height”, “ 503”); BAS. setParameter(“ worldid”,” 107″); You’ll have to figure out the rest, you’ll also need to set a world constant some how, you can use setParameter if you wish. (If you do this, your racing ahead, so set some static world address for now, later on i’ll show you how to implement the “ world getter”).

Okay, I decided that it’d be better to introduce the loaderhack into this sectionSo before we do that, make your self a new class, a bit like this: Code: package com. fusionbot. bot;/\*\*\* @FusionBot by ownagesbot\* BotAppletStub. java\* This code is not allowed to be used without explicit permission by the owner\* The code is wrote purely for educational purposes\* All usage must fully credit the owner\*/public class RSLoader extends ClassLoader {public RSLoader(String IP, String archive) {}}Then add yourself a loader reference to Bot, something like RSLoader loader = new RSLoader(IP, archive); archive is the name of the RS jar, you’ll need to get this whilst your navigatingPart B: The loader hack basic constructionNow you’ve got your reference to the loader, it’s time to build the class. Okay. Let’s get started.

You’ll want to get the jar first. So.. declare youself jar loaderJar; Now seeing as RS thinks you’ve navigated to that page, its only normal to get the jar, thats how we’ve overcome this kind of detection.

So, in your classes constructor use a JarURLConnection to download: “ jar: http://” + IP + “/” + archive + “!/” Another hint is to simply cast a url. openStream(); to JarURLConnectionThen to polish it off do a simple: loaderJar = jarStream. getJarFile(); Next: make this method in your class: Code: public Class;?; loadClass(String cName) throws ClassNotFoundException { return null; }Leave it blank for now. Make yourself a new class: Code: public class BotClassLoader extends ClassLoader {public BotClassLoader() {}protected final Class;?; defineClassSub(String name, byte[] b, int off, int len) { return super. defineClass(name, b, off, len); }protected final Class;?; findSystemClassSub(String name) throws ClassNotFoundException {return super.

findSystemClass(name);}This will be where all the classloading is done. You now have the basic loader structure, Section 3 will cover the rest of the loader hack. Part C: Finalising the GUI and the general Bot. java structureCode: app = (Applet) loader.

loadClass(code. replaceAll(“. class”, “”)). newInstance(); frame = new JFrame(“ My Bot – OwnageSBoT”); frame. setDefaultCloseOperation(JFrame. EXIT\_ON\_CLOSE); frame.

pack(); //pack the frame, this is mandatory, just puts it all together hard to explainframe. setSize(771, 533); //sets the frame sizeframe. setResizable(false); //disallows you to resizeapp. setStub(BAS); //Ok, so we just tell the applet to use BASapp.

setBackground(Color. white); frame. add(“ Center”, app); //Add the applet to the frame (from the center)frame. setVisible(true); //Make the frame visibleapp. init(); app.

start(); Once again, huge appologies for the indenting. Okay, basically what this does now, is cast “ app” (which I forgot to mention needs declaring in the class block), to an Applet of a new Instance of the loader. class. For those that don’t know variable code, is the name of the loader. class, which coinsidently also has to be found during the HTML parsing.

(All we do is replace “. class” with nothing so the class loader can load it)Note: this won’t load yet we haven’t sorted the loader hackThe next is just the GUI, you can copy paste or make some sexy GUI of your own. Section 3: An indepth look and completion of the loader hackPart A: The theoryOk, so the theory is that if we replace the class files that are loaded at runtime (such as client. class ka. class or w/e) by the loader, with our files, then we technically have bypassed their loadercheck, and effectively ran hacked classes.

The loader hack works like this: loader -> loads class java. lang. ClassLoaderwe replace java. land.

ClassLoader with our class: loader -> our class -> loads class java. lang. ClassLoadernow you see that by adding in our “ middle class” we have full control of what is sent to the java. lang. ClassLoader, therefore, if we have some “ pre-transformed” classes we can substitute them in. In this loader hack we will also inject “ getApplet()” this allows us to get the current applet instance thus later on we can get things like playerPos() in our accessormethods.

Part B: New class and constructor, with pseudocode and examplesYou may have done this earlier, if not it doesn’t matter, but you will need to create the second class mentioned earlier (BotClassLoader) and then just follow these steps. If you have, this is worth reading too. Okay so: To start with add yourself a new class, I called mine RSLoader. Add a constructor and lets get started.

Now, you may have noticed that when navigating through the pages, we don’t download the loader, well, we do… now . So lets get started, if you know how to do this, go ahead and make your own method and skip this part. Firstly declare a global jarfile, I called mine loaderJar.

Code: private JarFile loaderJar; Next, we’ll use a jarurlconnection, and get the jar. So.. something like this would be appropriate: Code: public RSLoader(String IP, String archive) throws IOException {System. out.

print(“ Attempting to download the loader..”); URL u = new URL(“ jar: http://” + IP + “/” + archive + “!/”); JarURLConnection jurl = (JarURLConnection) u. openConnection(); loaderJar = jurl. getJarFile(); System.

out. println(“ Done”);}As you can see, we create a new constructor with 2 parameters, archive (the jar’s name), and IP (the world URL)we then create a new url, and cast it to a jarurlconnection, this allows us to get the jar file and set out loaderJar. Part C: The masterhack behind it allSorry about the previous part, it was a kind of repeat, but never the less, it’s all good to take in. To ensure your up with me, i’ve wrote a checklist: You should now have 4 classes: Bot, BotAppletStub, RSLoader and some kind of SpoofBrowser class.

BotAppletStub should now be a complete AppletStub. RSLoader should extend java. lang. ClassLoader, it should now obtain the runescape jar and set loaderJar. Bot should create a new RSLoader instance and should be cast into an Applet.

If you have done all this, proceed otherwise go back and fill in the stuff you missed. Okay, now you may or may not have overidden loadClass yet, so i’ll include it anyway. Code: public Class;?; loadClass(String cName) throws ClassNotFoundException { }Alright, so now when we loadClass is called, it’ll go through our loadClass (don’t forget we call it to load loader). Now for the hack. Code: Class c = nulltry {c = super. findSystemClass(cName);}catch(ClassNotFoundException CNFE) {//Our code is going to go here}catch(Exception e) { e.

printStackTrace(); } //Ofkif(c != null) return c; Okay, so all this does is: Set variable c, and initialize it to null. It then tries to find the class (it will only be found if it’s been previously loaded), and set c to that class if it is. If not, it’ll launch our hack on the class. And then we have a basic exception catch for any other errors. If c was found it wouldn’t be null, it’d be the class, thus if c != null then a class was found, return that class.

Replace //Our code is going to go here with this: Code: ZipEntry zip = loaderJar. getEntry(cName. replaceAll(“.”, “/”) + “. class”); ClassGen cg = new ClassGen(new ClassParser(loaderJar. getInputStream(zip), zip.

getName()). parse()); int classidx = ((ConstantClass) cg. getConstantPool(). getConstant(cg.

getSuperclassNameIndex())). getNameIndex(); ConstantUtf8 utf = ((ConstantUtf8) cg. getConstantPool(). getConstantPool(). getConstant(classidx)); This firstly gets the class loaded out of the jar into a zipentry.

The zipentry is then parsed through a ClassParser. A classgen is then constructed from the ClassParser. The super class name index is then found. We then make a utf from the index. This means we can check to see what the superclass’s name actually is.

The next stage then, is to check if its java. lang. ClassLoader, if it is, we want to replace it with our hack. Underneath add this code: Code: if(“ java/lang/ClassLoader”. equals(utf.

getBytes())) {utf. setBytes(“ BotClassLoader”); int count = 0; for(Constant con : cg. getConstantPool(). getConstantPool(). getConstantPool()) {if(con != null) {if(con instanceof ConstantMethodref) {ConstantMethodref cm = (ConstantMethodref) con; ConstantNameAndType cnt = (ConstantNameAndType) cg.

getConstantPool(). getConstantPool(). getConstant(cm. getNameAndTypeIndex()); ConstantUtf8 name = ((ConstantUtf8) cg. getConstantPool(). getConstantPool().

getConstant(cnt. getNameIndex())); if(name. getBytes(). equals(“ defineClass”) || name.

getBytes(). equals(“ findSystemClass”)) {count++; name. setBytes(name. getBytes() + “ Sub”);}if(count > 1) break;}}}}Okay, so if utf.

getBytes() equals “ java/lang/ClassLoader” proceed with the hack. (Basically if this class extends java. lang. ClassLoader proceed)Next replace java.

lang. ClassLoader with BotClassLoader, our own class loader. Now everything will go through our class. Now, we’ll loop through all the constants in the constantpool and if they don’t equal null, and are an instanceof ConstantMethodref (basically if they refer to a method) then proceed.

We then go through a couple of steps to get the name of the method. If the name is “ defineClass” or “ findSystemClass” then change there name to: name + “ Sub”. Which suprisingly..

is the name of our methods . Now we could call the loader hack done, but it’s not yet (we haven’t quite finished this process actually), we have some more little hacks to do, whilst we’re in the loader we may as well do all loader related things. Theres 3 to do: 1. The ClassLoader “ override”. 2. Add a new method that lets us “ getClient” (which is the clients applet instance).

3. Add the loader interface (This ensures that the methods are there, in this case it’ll be used for “ getClient”). So the next step is the getClient: underneath add this: Code: for(Field f : cg. getFields()) {if (f.

getType(). equals(new ObjectType(“ java. applet. Applet”))){cg = injectObject(cg,” java. applet.

Applet”, f. getName(),” getClient”); break;}}Okay, so we loop through all the fields, and if one of the fields is of type “ Applet” then we’ve got the field that contains the RSApplet instance. We then inject a new object (or a new field(or is it a method I cant remember) of type Applet. This will basically equal w/e the field name was, so it’d be either: Code: Applet getClient = FIELD\_NAME; orCode: Applet getClient { return FIELD\_NAME; }I can’t remember which but either way it doesn’t matter. That returns a classgen, and we set our current one to the new modified one.

Here is injectObject which I won’t comment on (it was bcelified). Code: public ClassGen injectObject(ClassGen cg, String type, String varName, String methodName) {ConstantPoolGen cp = cg. getConstantPool(); InstructionFactory ifactory = new InstructionFactory(cg, cp); InstructionList ilist = new InstructionList(); MethodGen method = new MethodGen(Constants. ACC\_PUBLIC, new ObjectType(type), Type.

NO\_ARGS, new String[] {}, methodName, cg. getClassName(), ilist, cp); ilist. append(ifactory. createLoad(Type.

OBJECT, 0)); ilist. append(ifactory. createFieldAccess(cg. getClassName(), varName, new ObjectType(type), Constants.

GETFIELD)); ilist. append(ifactory. createReturn(Type. OBJECT)); method.

setMaxStack(); method. setMaxLocals(); cg. addMethod(method. getMethod()); ilist. dispose(); return cg;}Ah..

it is a method, oh well I cba to remove what I typed. The next thing to do is add an interface that allows us to use this method even though its not injected until runtime. So create a new folder, call it interfaces (or hooks if you want to use that (MS did)). create a new interface, I called mine intLoader, heres what you need in it: Code: package interfaces; import java. applet. Applet; public interface intLoader {public Applet getClient();}Then underneath the getClient hack in loadClass add this: Code: cg.

addInterface(“ interfaces. intLoader”); The next thing to do is “ defineClass” this means we can now make c our class, our modified loader. so add this: Code: byte buffer[] = cg. getJavaClass(). getBytes(); c = defineClass(cName, buffer, 0, buffer. length); fairly simple, theres no need to understand it indepth anyway. One final thing to do, underneath your if(c != null) return c; add this: Code: super. loadClass(cName); Hopefully that will never ever be called, but basically if the class wasn’t found and our loader hack didn’t work properly and defineClass couldn’t define it, c won’t be returned, and the compiler demands \*something\* to be returned, so we just let the real classloader deal with it, and return the result of that. Onto section 4Section 4: Getting to a stage that will compile, and allow us to login etc. Part A: Building up your own frame and adding your applet to itAlright, at this point your bot should compile and run but do nothing (visually) and then end. So it’s time to build ourselves a frame and add all the stuff we need to it. Okay so under where you declare your applet declare a JFrame, I called mine frame but it’s up to you. Code: public static JFrame frame; Now under where you set the app, create a new frame so something like: Code: frame = new JFrame(“ FusionBot – OwnageSBoT”); will do just fine. Next you have a choice of what to do when the “ X” is pressed, heres mine: Code: frame. setDefaultCloseOperation(JFrame. EXIT\_ON\_CLOSE); Pack the frame: Code: frame. pack(); I actually think that theres no point doing it now, infact im not sure you need to pack at all but meh to it, doesnt break the bot so why notSet the size: Code: frame. setSize(771, 533); If your spicing up your frame and adding some special GUI features you’ll probably need a diff size iunnoStop resizing: Code: frame. setResizable(false); Okay, now we have the visuals working lets finish off the applet and add it to our frame!! Set the applet’s stub that we made earlier: Code: app. setStub(BAS); I liek a white bg so: Code: app. setBackground(Color. white); Swoot lets add the applet to the frame now then xDCode: frame. add(“ Center”, app); Make the frame visible: Code: frame. setVisible(true); Initialize and start the applet: Code: app. init(); app. start(); Compile it and run it, you should be able to login now