

Forum:[Hack Forums](#)

Thread: [Legion | Editing Java Class Files via Bytecode/Java Reverse Engineering](#)

lillix - October 30, 2013, 10:01 am

[img]http://i.minus.com/i6zNsXBs2f6V8.gif[/img]

Editing Java Class Files via Bytecode

Contents

- How Java code is stored
- Understand how to use the tools necessary to reverse engineer Java
- Understand how to exploit possible vulnerable security functions
- Testing what you've learned

How Java code is stored

Java is a programming language which is compiled into files called 'classes', non-readable to the human eye. A decompiled version of a class file is a Java file, which contains the source code used to operate the application. The most basic JAR (Java Archive) would contain the following files (example):Main.class

META-INF

As the name suggests, the file Main.class is a compiled Java file, which is also the entry point of the application - without the entry point, the application may not execute. The META-INF folder you may see is a file within a folder which contains instructions for the JAR to follow. It can include, but not limited to, the main class used. Identifying the entry point is vital if you want to bypass any authentication.

Understand how to use the tools necessary to reverse engineer Java

The only tools we will be using is [JD-GUI](#), the full term meaning "Java Decompiler Graphical User-Interface", allowing you to decompile class files down to understandable Java source. The decompilations aren't fully accurate, but still give you a basic understanding of how a piece of program works. We will also use [JBE](#), the full term meaning "Java Bytecode Editor", allows you to edit the machine code instructions of a certain class file from the Java application. You might want to explore the GUI of both applications before we start any reverse engineering.

Understand how to exploit possible vulnerable security functions

Java security systems are similar to that of VB.NET security systems, it will always return some type of value, obviously exploitable, under the right circumstances and user. A basic security system would send some type of request to a server, checking if a key or any other likewise 'token' is correct - in the case that the token is correct, a value is returned. e.g. 1 - in the case that the token is invalid, a different value is returned, such as 0, or 2; any value that is significantly different to that of the correct value. Not many developers take the time to come up with innovative methods of protecting their applications, and thus are lazy, a good thing for us, a bad thing for the customer(s) of the application.

Testing what you've learned

Assuming you have already proceeded to explore applications that I have supplied prior to the

previous part, we'll dive straight into the reverse engineering. You may want to use a crackme from <http://crackmes.de/> which I highly recommend. To download a crackme, you'll have to register. Once registered, click 'Advanced search', under the basic search text box, then select 'Java' from the drop menu, and select a difficulty you'd like - for the sake of the tutorial, I'll download a level two difficulty file, from here: http://crackmes.de/users/vhly/java_crackme_3/download

Download it and unzip it, then proceed to open the JAR file with JD-GUI. Once opened, a list-view will appear on the far left column, and what looks like brown posting boxes are something called 'Packages'. These are used to organize program source and store Java files, which will be later compiled into class files. To find the entry point of the application, expand packages that are not META-INF, until you find a class name which resembles an entry-point of some type. Click the class name found, and you will notice the white space in the center fill up with some Java source from the main class. Inside the class, search for the text 'static void main(String[] args)', and if you can find this, you have found the entry point. If not, keep searching. Once you stumble across that specific text, look for some type of error messages that may popup. This is my security method:

All I have to do is remove the sections of the function that exit the application upon failure of authentication, which is the function named 'l', inside the class named 'l', and if you look inside this function, you'll see something that resembles the following:

So to do this, you need to open JBE, which I provided a download link for in the second part. Open the JAR in an archiving tool, and extract the entry point class to somewhere you can easily access. Now open JBE, and click the browse button, and find the entry point class that you extracted. Once opened, expand the 'Methods' tree on the left menu. Once that's expanded, expand the folder which says 'main', now click on '[0] Code', inside that folder. Now to your right, you'll see the bytecode of that class file load up. You can look through it, but what we want is the source editor, so click that tab, then scroll to the top. This is the hardest part - but since strings are loaded into memory on separate lines, it'll be easier to identify where the checks are, and where error dialogs are. You need to find something that resembles 'invokestatic com/vhly/crackmes/CM3/I()V', this may differ on the crackme you are using. If you find that text, you want to remove that line and each occurrence of it. Once removed, click 'Save method', then drag the saved file into the JAR. Open the JAR back up in JD-GUI, then look at the main class. All exit occurrences are now removed. It still isn't cracked though - the aim of this tutorial isn't to crack anything, but to show you the opportunities of JBE. Some bytecode references can be found online.

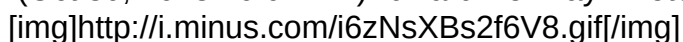
Need further help? PM me please. ?
Thanks for taking the time to read this.

Nikhufu - November 9, 2013, 9:09 am

Thanks for the tutorial. I'm new at this stuff and this tutorial helped me understand the basics.

The Old Worm - November 9, 2013, 9:47 am

(Oct 30, 2013 10:01 AM)Edward Kenway Wrote: [\[CITE\]](#)

[img]

Editing Java Class Files via Bytecode

Contents

- How Java code is stored
- Understand how to use the tools necessary to reverse engineer Java
- Understand how to exploit possible vulnerable security functions
- Testing what you've learned

How Java code is stored

Java is a programming language which is compiled into files called 'classes', non-readable to the human eye. A decompiled version of a class file is a Java file, which contains the source code used to operate the application. The most basic JAR (Java Archive) would contain the following files (example):Main.class

META-INF

As the name suggests, the file Main.class is a compiled Java file, which is also the entry point of the application - without the entry point, the application may not execute. The META-INF folder you may see is a file within a folder which contains instructions for the JAR to follow. It can include, but not limited to, the main class used. Identifying the entry point is vital if you want to bypass any authentication.

Understand how to use the tools necessary to reverse engineer Java

The only tools we will be using is [JD-GUI](#), the full term meaning "Java Decompiler Graphical User-Interface", allowing you to decompile class files down to understandable Java source. The decompilations aren't fully accurate, but still give you a basic understanding of how a piece of program works. We will also use [JBE](#), the full term meaning "Java Bytecode Editor", allows you to edit the machine code instructions of a certain class file from the Java application. You might want to explore the GUI of both applications before we start any reverse engineering.

Understand how to exploit possible vulnerable security functions

Java security systems are similar to that of VB.NET security systems, it will always return some type of value, obviously exploitable, under the right circumstances and user. A basic security system would send some type of request to a server, checking if a key or any other likewise 'token' is correct - in the case that the token is correct, a value is returned. e.g. 1 - in the case that the token is invalid, a different value is returned, such as 0, or 2; any value that is significantly different to that of the correct value. Not many developers take the time to come up with innovative methods of protecting their applications, and thus are lazy, a good thing for us, a bad thing for the customer(s) of the application.

Testing what you've learned

Assuming you have already proceeded to explore applications that I have supplied prior to the previous part, we'll dive straight into the reverse engineering. You may want to use a crackme from <http://crackmes.de/> which I highly recommend. To download a crackme, you'll have to register. Once registered, click 'Advanced search', under the basic search text box, then select 'Java' from the drop menu, and select a difficulty you'd like - for the sake of the tutorial, I'll download a level two difficulty file, from here: http://crackmes.de/users/vhly/java_crackme_3/download

Download it and unzip it, then proceed to open the JAR file with JD-GUI. Once opened, a list-view will appear on the far left column, and what looks like brown posting boxes are something called 'Packages'. These are used to organize program source and store Java files, which will be later compiled into class files. To find the entry point of the application, expand packages that are not META-INF, until you find a class name which resembles an entry-point of some type. Click the class once found, and you will notice the white space in the center fill up with some Java source from the main class. Inside the class, search for the text 'static void main(String[] args)', and if you can find this, you have found the entry point. If not, keep searching. Once you stumble across that specific text, look for some type of error messages that may pop up. This is my security method:

All I have to do is remove the sections of the function that exit the application upon failure of authentication, which is the function named 'I', inside the class named 'I', and if you look inside this function, you'll see something that resembles the following:

So to do this, you need to open JBE, which I provided a download link for in the second part. Open the JAR in an archiving tool, and extract the entry point class to somewhere you can easily access. Now open JBE, and click the browse button, and find the entry point class that you extracted. Once opened, expand the 'Methods' tree on the left menu. Once that's expanded, expand the folder which says 'main', now click on '[0] Code', inside that folder. Now to your right, you'll see the bytecode of that class file load up. You can look through it, but what we want is the source editor, so click that tab, then scroll to the top. This is the hardest part - but since strings are loaded into memory on separate lines, it'll be easier to identify where the checks are, and where error dialogs are. You need to find something that resembles 'invokestatic com/vhly/crackmes/CM3/I/I()V', this may differ on the crackme you are using. If you find that text, you want to remove that line and each occurrence of it. Once removed, click 'Save method', then drag the saved file into the JAR. Open the JAR back up in JD-GUI, then look at the main class. All exit occurrences are now removed. It still isn't cracked though - the aim of this tutorial isn't to crack anything, but to show you the opportunities of JBE. Some bytecode references can be found online.

Need further help? PM me please. ?

Thanks for taking the time to read this.

Your tutorial is not even useful.

Code iconst_0

ireturn