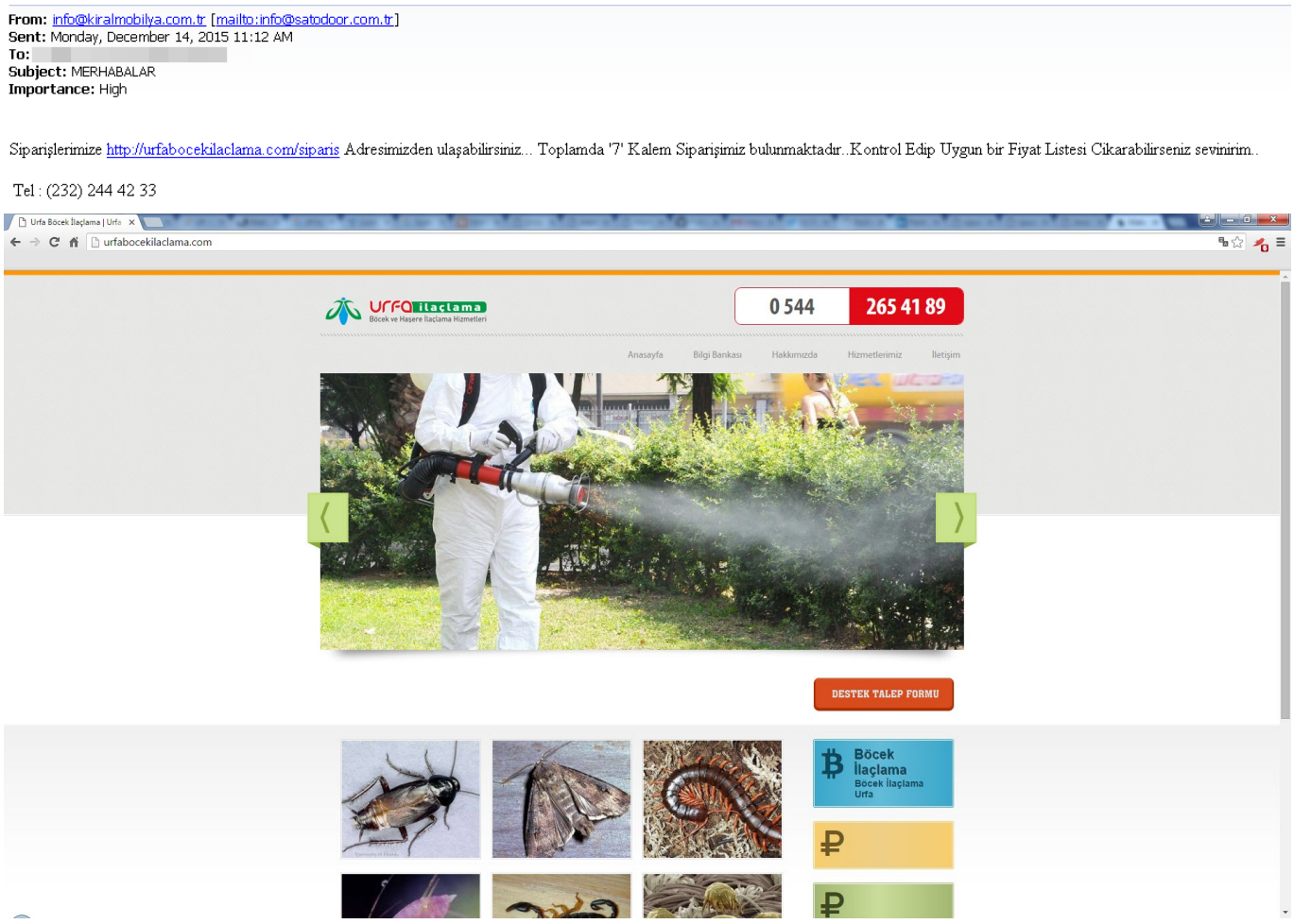
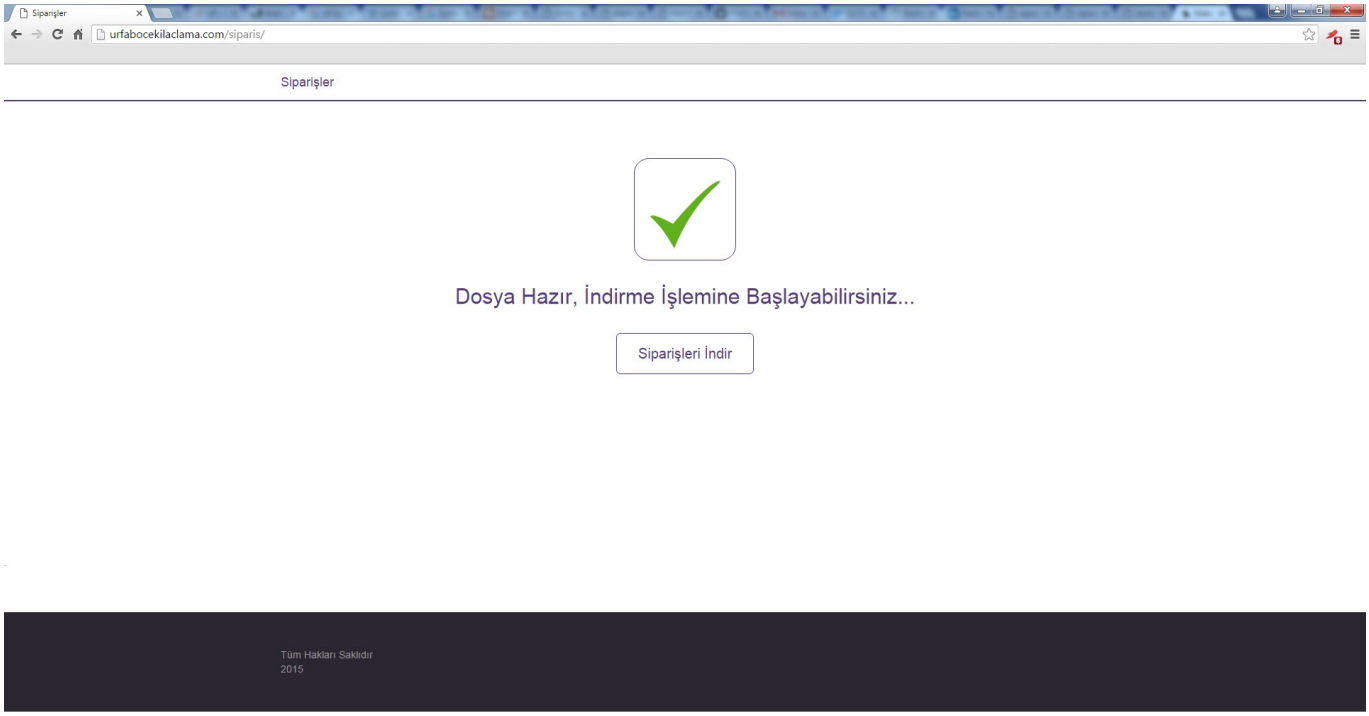
Java Decompilers

written by Mert SARICA | 1 March 2016

I agree that working at byte code level is sometimes a bit challenging. If the mission is analyzing a Java malware, decompiling the class files into Java source code is the first step most analysts would take. However, like I mentioned in my post on July (Java Byte Code Debugging), if you are up against a malware that takes advantage of an obfuscator tool (like Allatori) Java decompilers (like JD) can most of the time let you fail.

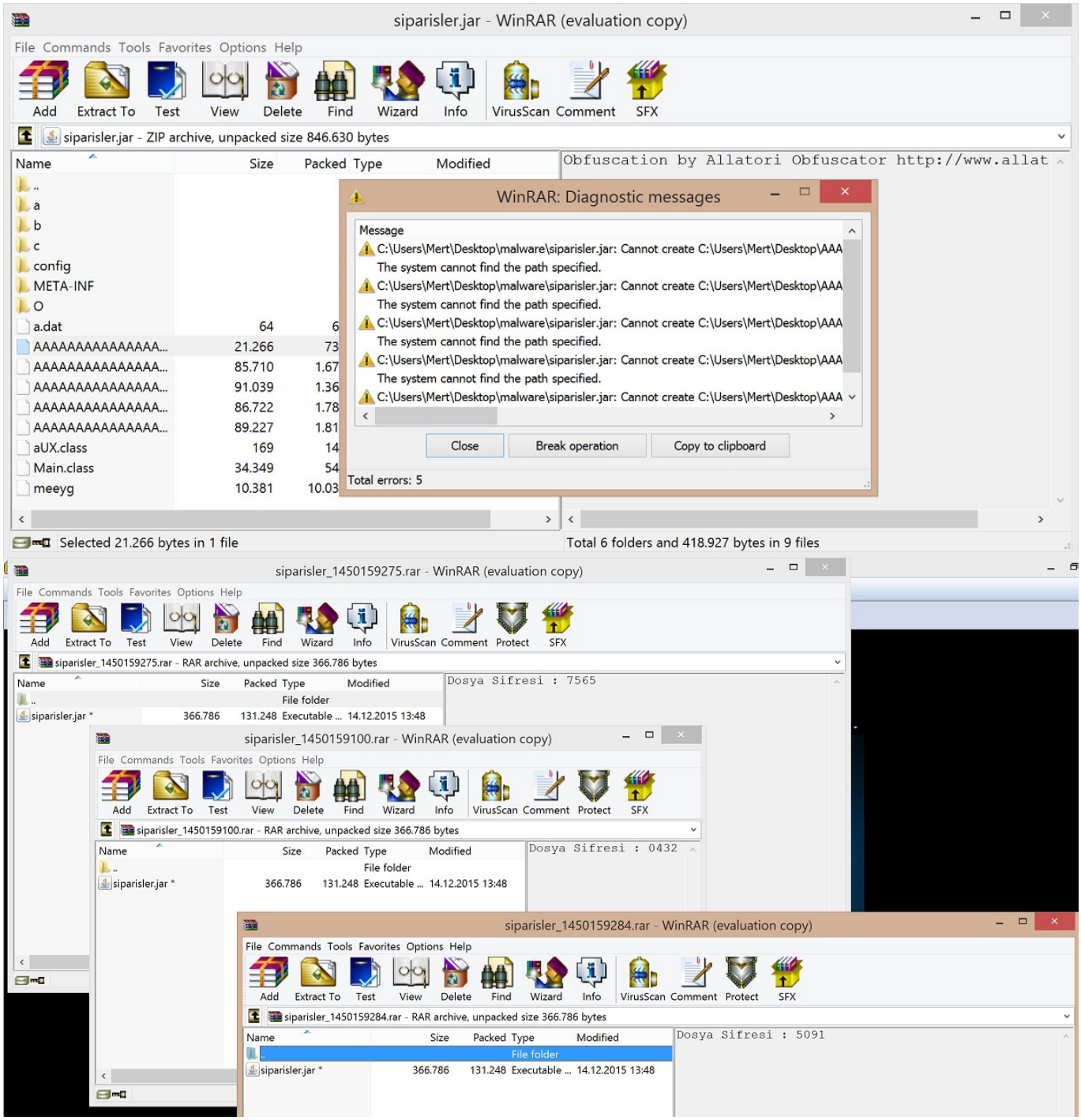
Around December, a Java malware called siparisler.rar (siparisler.jar) took my attention which downloads additional payload from a website that I think was hacked with different password each time.





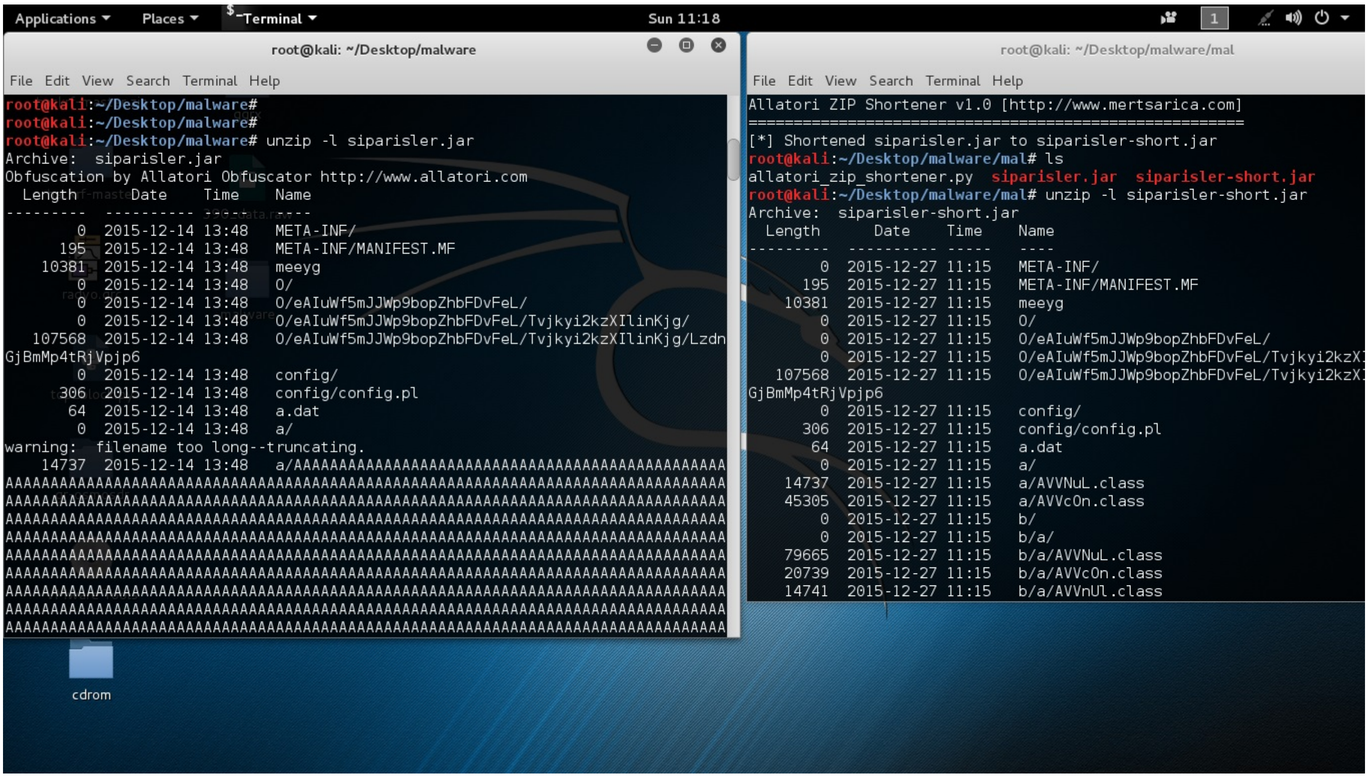
When I looked at the class files inside the JAR package that was made with the help of Allatori obfuscator tools' strongest features (long class and method names, reserved names like AUX e.t.c), I saw that file names were approximately 8000 digits long.

Because of the long file names, when I tried to extract the malware to the operating system with the use of tools like Winrar, 7zip, Unzip I realized that I got stuck at operating system limits and was not able to open the files. Also because of the long file and method names I noticed that most decompilers (except CFR) got an error during the decompilation process.

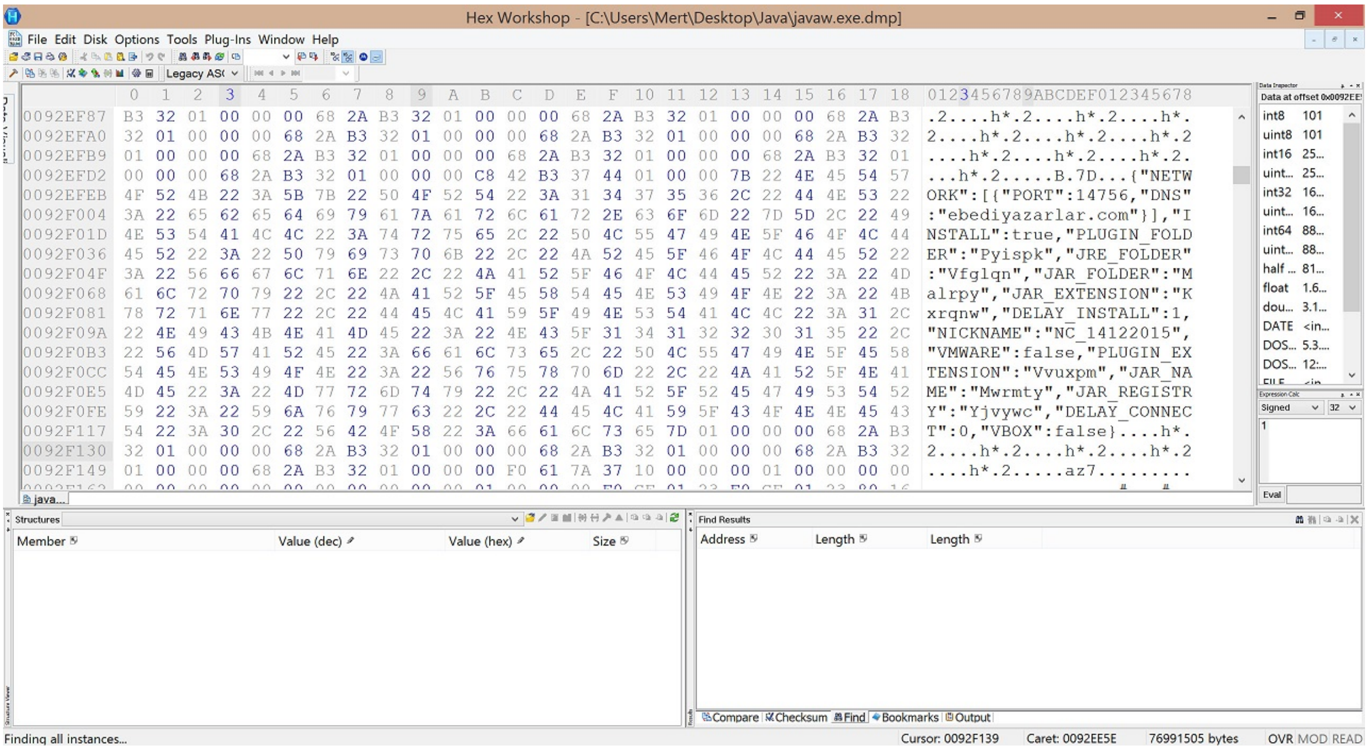


As a person who witnessed that this malicious file was crashing a commercial product that does sandbox analysis while it is analyzing, I can state again that corporations whom only invest in and rely upon devices are on a thin ice.

Of course, with Python a simple tool like Allatori Zip Shortener, making this zip file openable was easy enough.



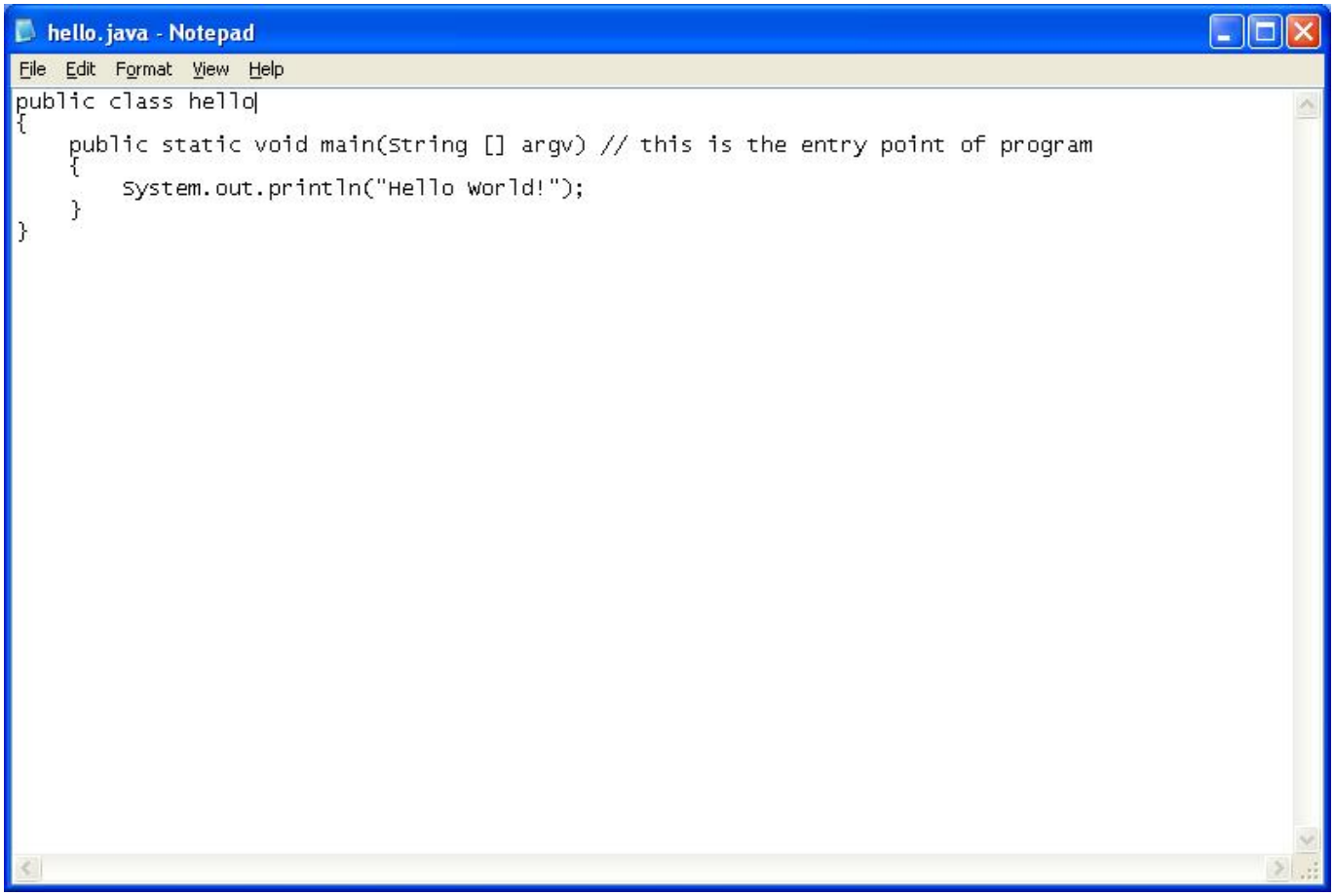
To gain a more detailed information, I could have continued with a byte code level analysis, like in the article that I posted on July. If my purpose was to only find the type of this malware, I could have learned that it is a Jsocket RAT software by doing a quick search on the memory file without any trouble.



While I was analyzing this malware, I realized that most decompilers were not enough against Allatori thanks to a website

(http://www.javadecompilers.com/).Most decompilers were either unsuccessful to decompile this malware into a source code or the source code they decompiled was in a condition that could not be reorganized. If I wanted to move on with a byte code level static analysis, I could have seen that Allatori is hiding the strings and to be able to solve it I would have needed to find the method of hiding which would have made my work longer at byte code level. Because of this, I decided to evaluate existing decompilers against Allatori and try to find out which decompiler can reveal the algorithm that was used for hiding those strings. The criterion for success was that the class file which got decompiled to source code was reorganizable and executable.

First of all, in Java I wrote a simple code which prints “Hello World” to the command line and compiled it into a JAR package. Then with the Allatori, I create an obfuscated HelloWorld package. Finally, I started to decompile all the JAR files into the source code and then compile and run them by using this site http://www.javadecompilers.com/



config.xml - Notepad

File Edit Format View Help

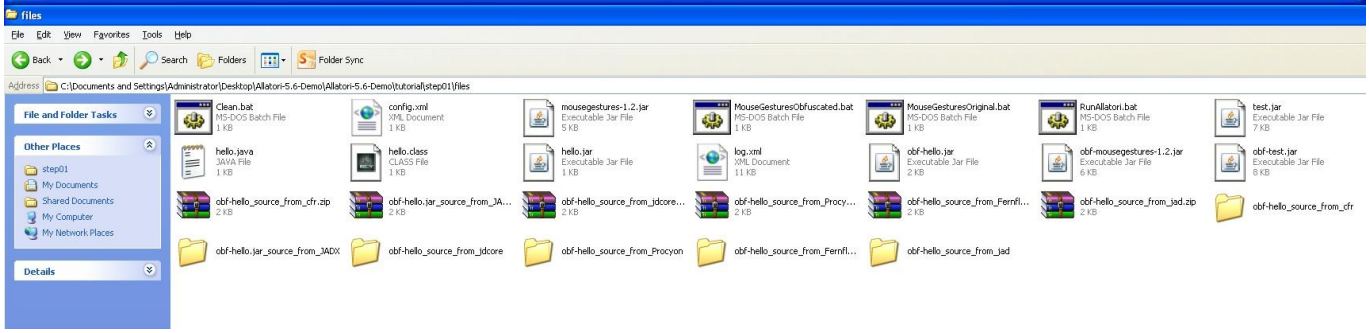
```
<config>
    <input>
        <jar in="hello.jar" out="obf-hello.jar"/>
    </input>

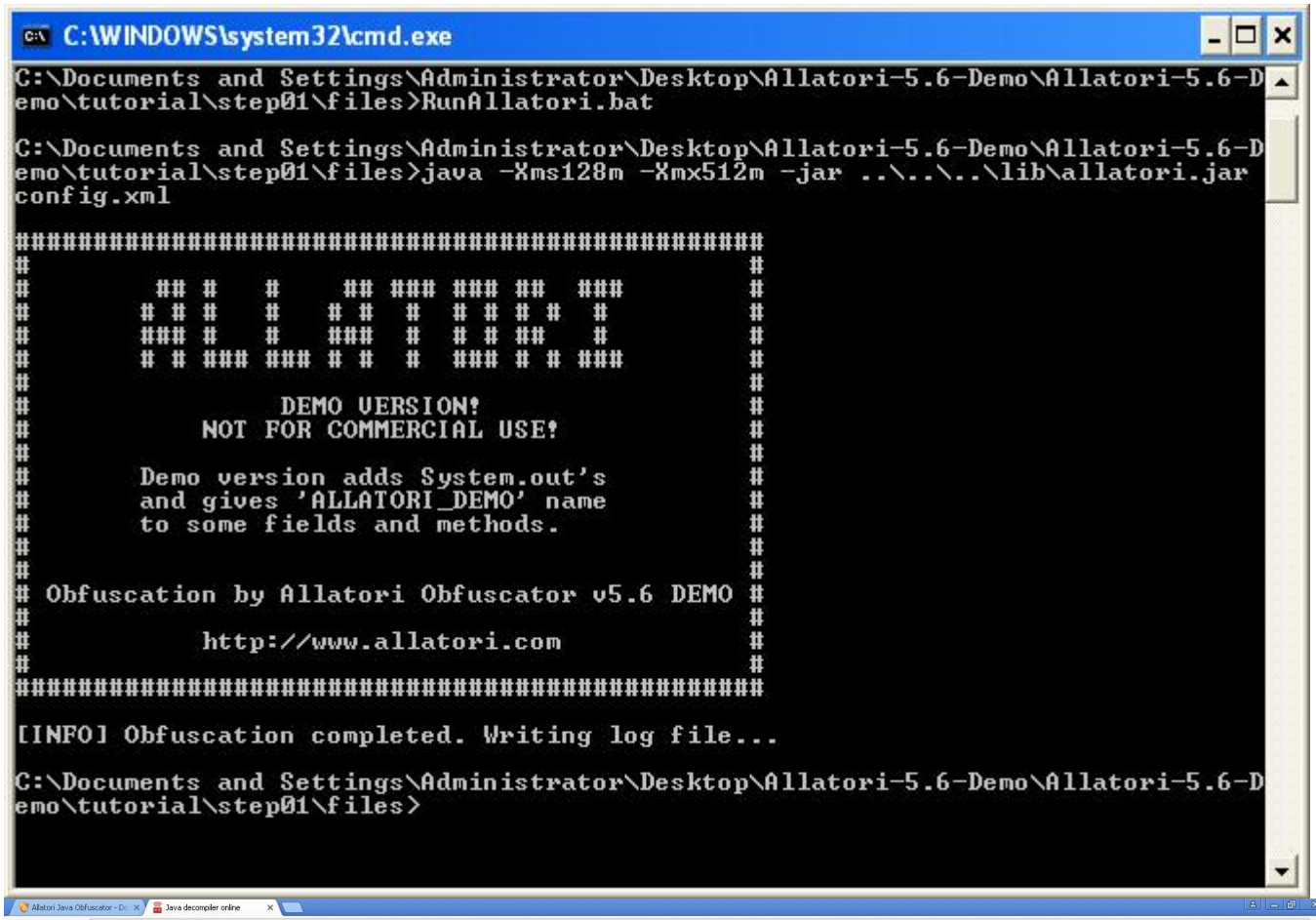
    <keep-names>
        <class access="protected+">
            <field access="protected+"/>
            <method access="protected+"/>
        </class>
    </keep-names>

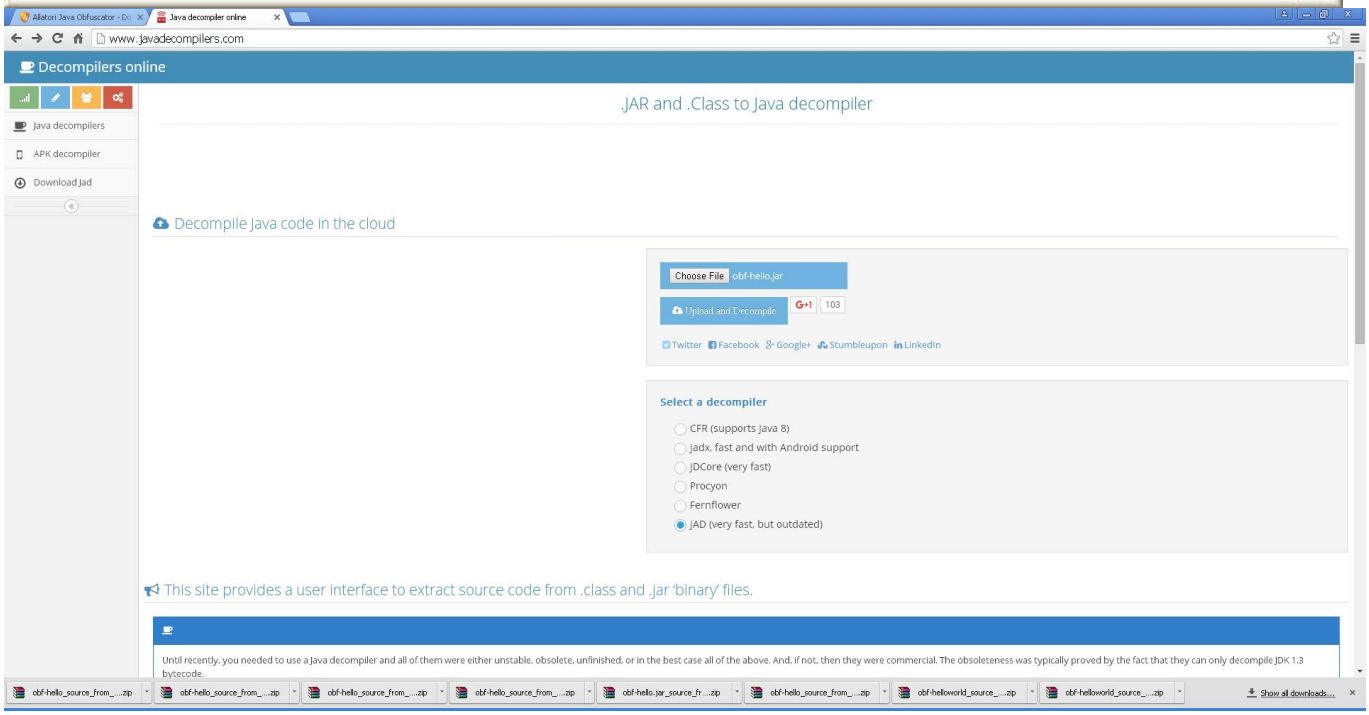
    <!-- String encryption -->
    <property name="string-encryption" value="maximum"/>
    <property name="string-encryption-type" value="strong"/>
    <property name="string-encryption-version" value="v4"/>
    <!-- <property name="string-encryption-ignored-strings" value="patterns.txt"/> -->

    <!-- Control flow obfuscation
    <property name="control-flow-obfuscation" value="enable"/>
    <property name="extensive-flow-obfuscation" value="maximum"/>
    -->

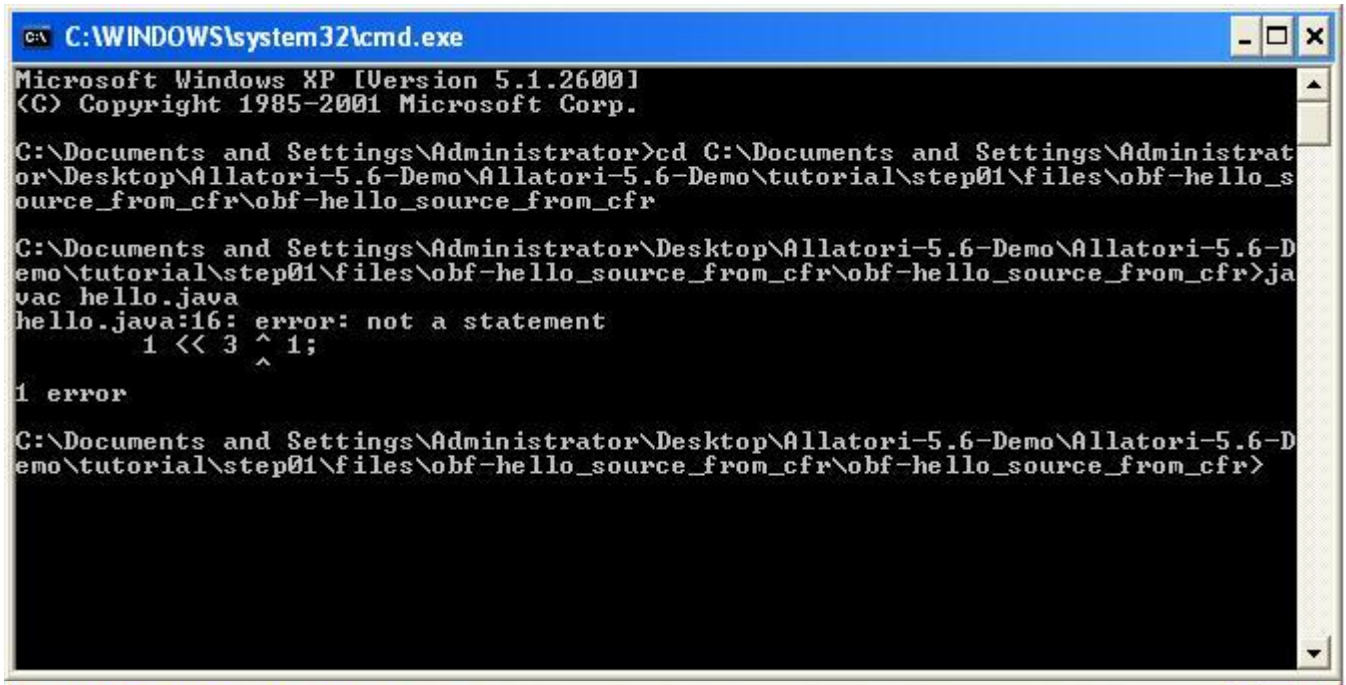
    <property name="log-file" value="log.xml"/>
</config>
```

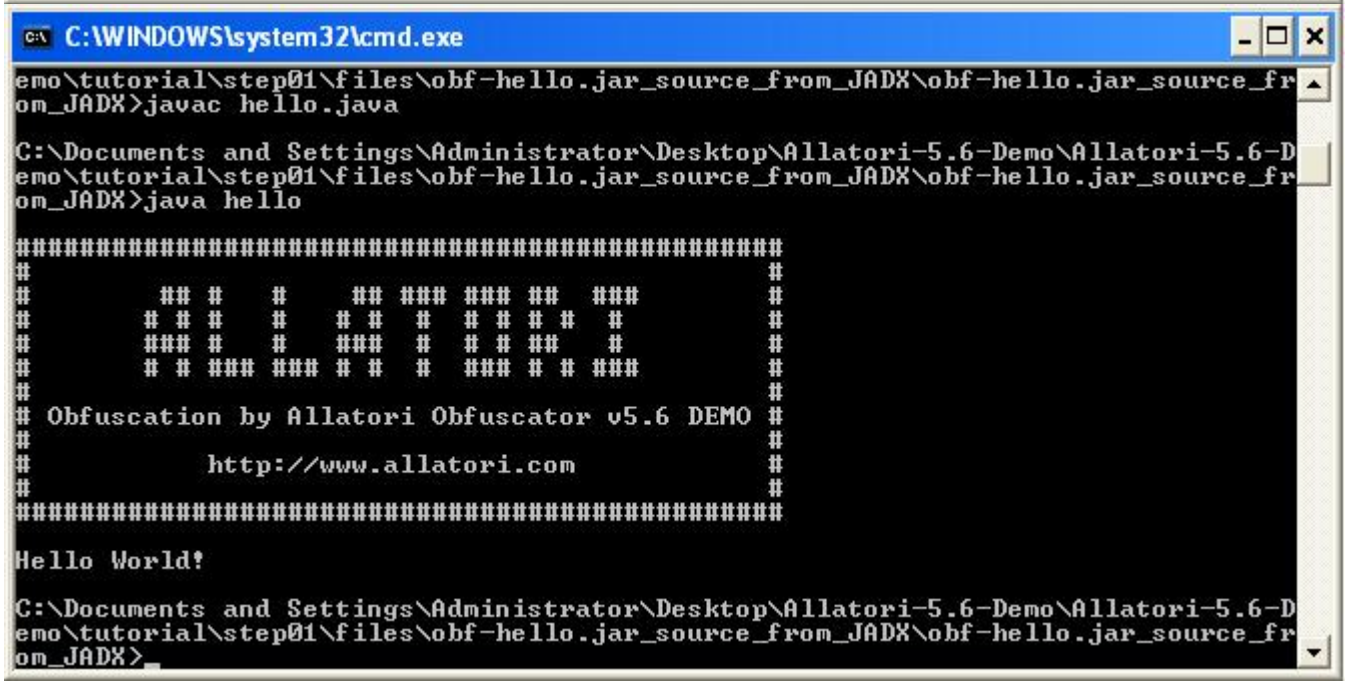


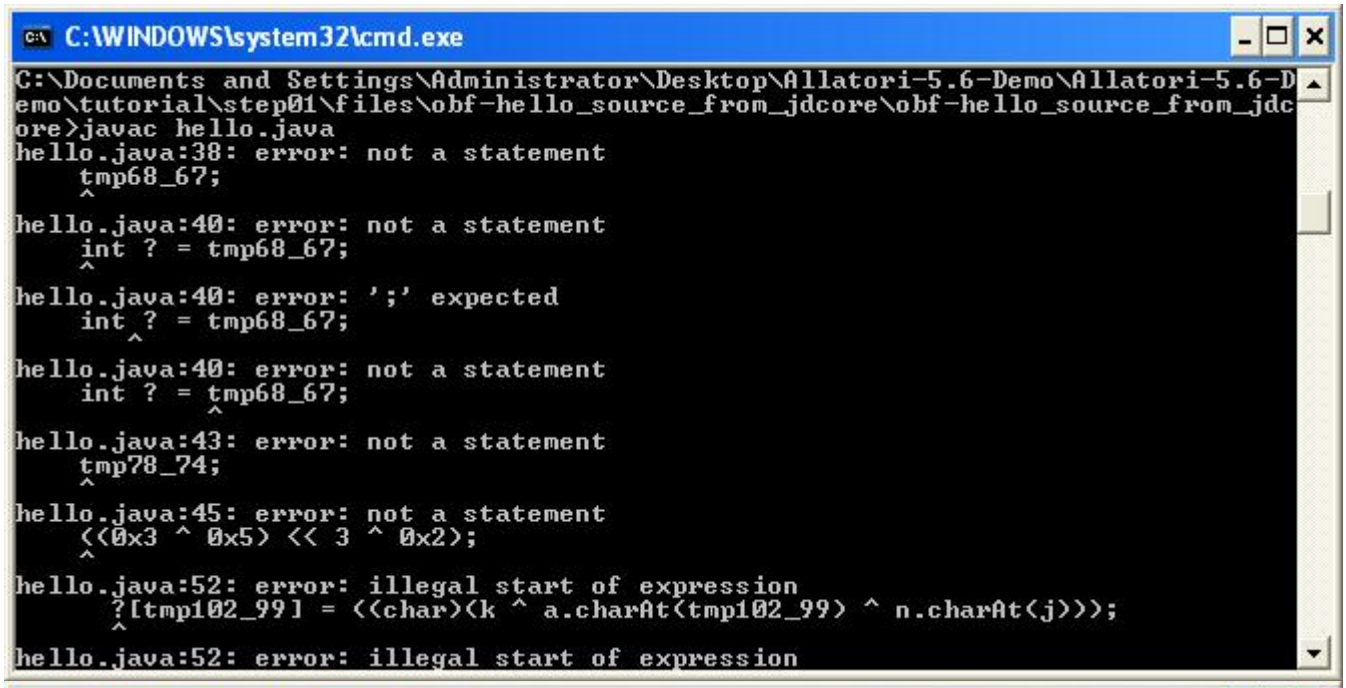


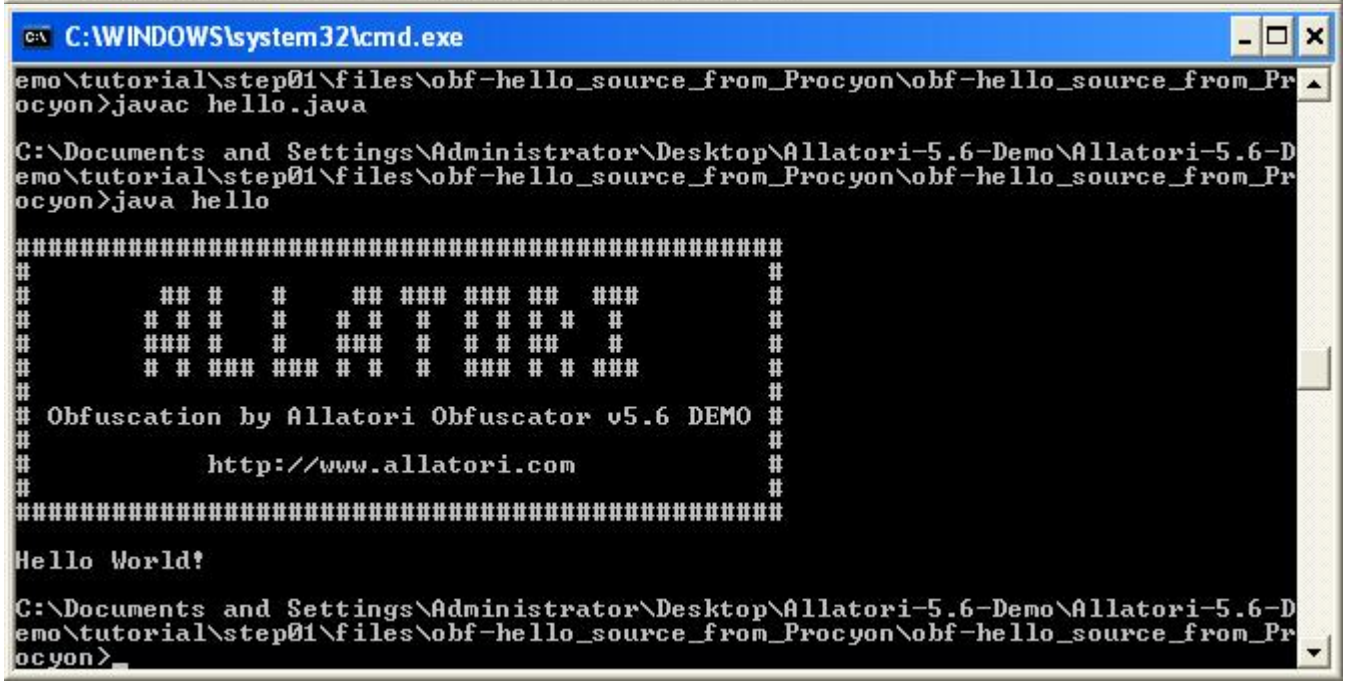


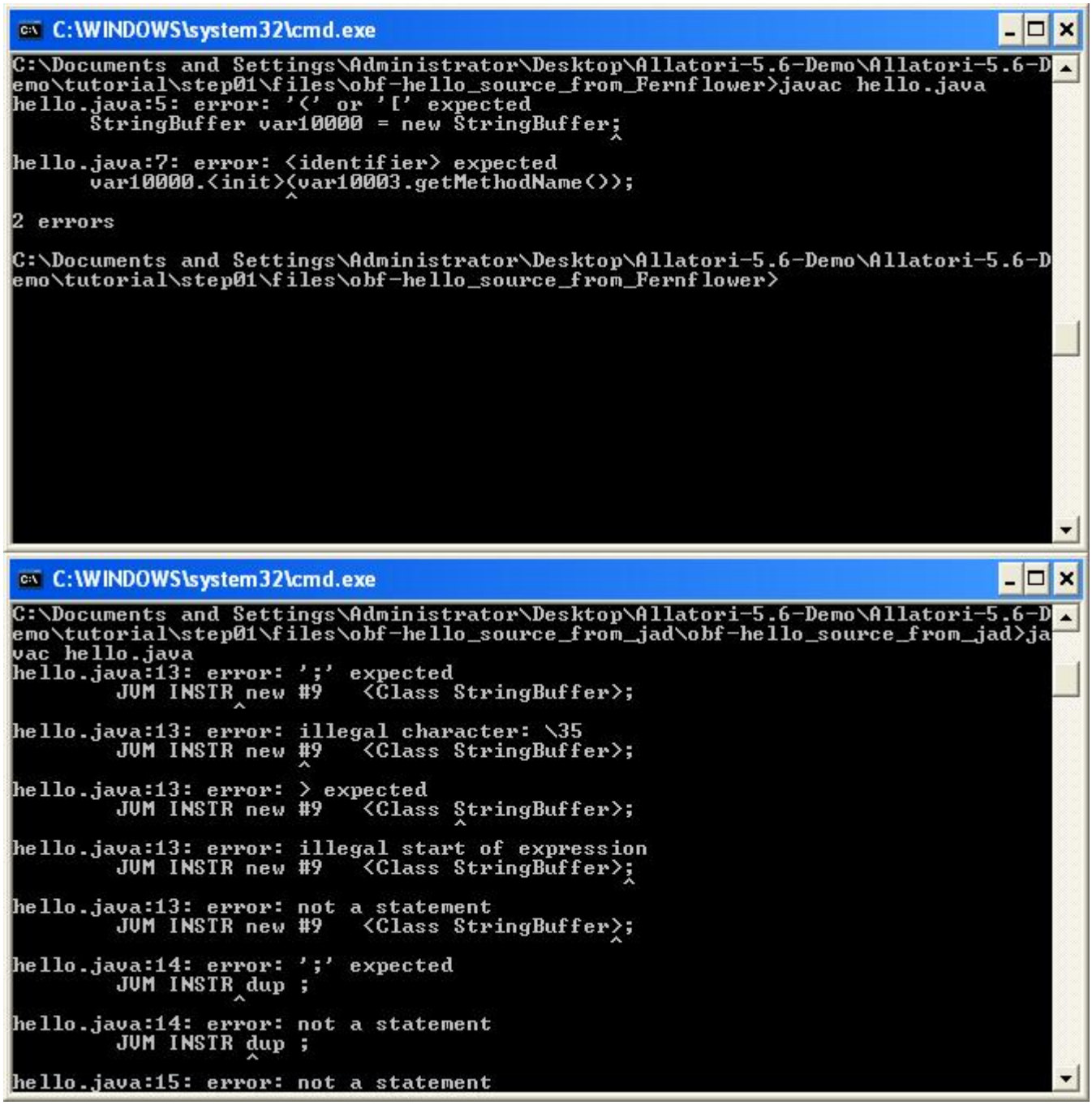
As a result of the evaluation I found out that JadX and Procyon decompilers were able to successfully decompile the codes that were hidden by Allatori v5.6 Demo version to their original form.


C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator>cd C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-Demo\tutorial\step01\files\obf-hello_source_from_cfr\obf-hello_source_from_cfr
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-Demo\tutorial\step01\files\obf-hello_source_from_cfr\obf-hello_source_from_cfr>javac hello.java
hello.java:16: error: not a statement
1 << 3 ^ 1;
1 error
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-Demo\tutorial\step01\files\obf-hello_source_from_cfr\obf-hello_source_from_cfr>

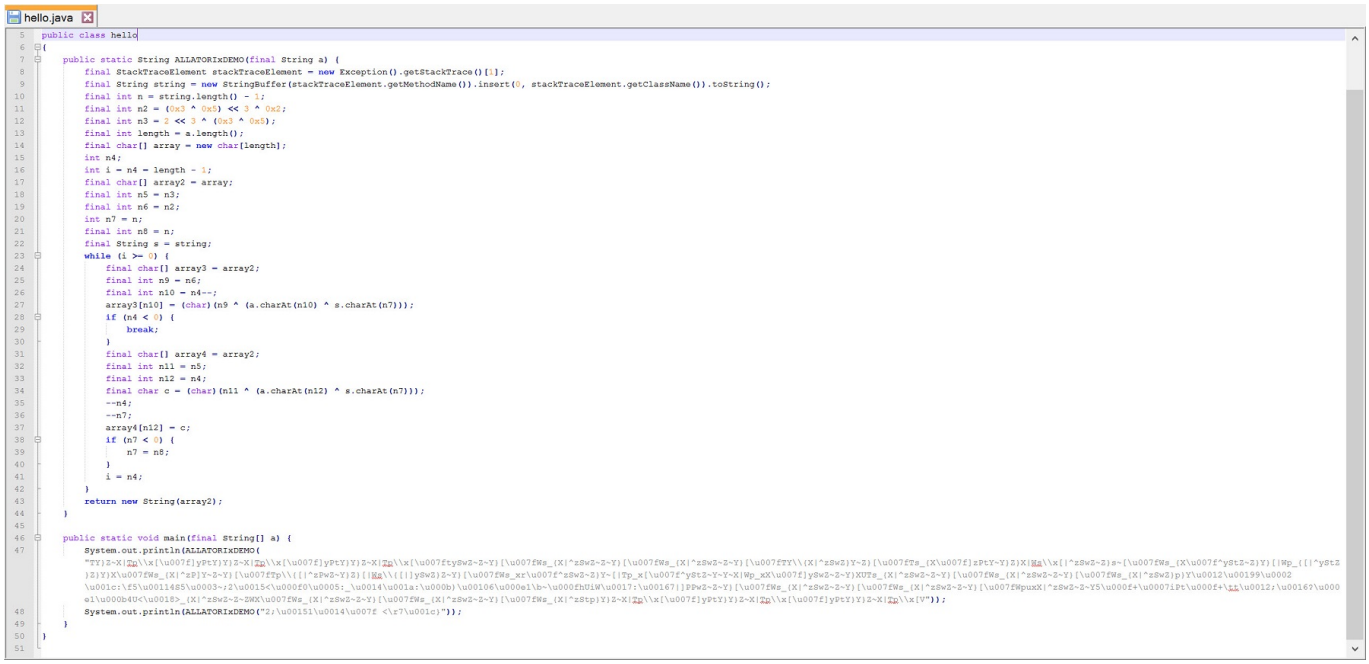

C:\WINDOWS\system32\cmd.exe
emo\tutorial\step01\files\obf-hello.jar_source_from_JADX\obf-hello.jar_source_from_JADX>javac hello.java
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-Demo\tutorial\step01\files\obf-hello.jar_source_from_JADX\obf-hello.jar_source_from_JADX>java hello
Obfuscation by Allatori Obfuscator v5.6 DEMO
http://www.allatori.com
Hello World!
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-Demo\tutorial\step01\files\obf-hello.jar_source_from_JADX\obf-hello.jar_source_from_JADX>


C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-D
emo\tutorial\step01\files\obf-hello_source_from_jdcore\obf-hello_source_from_jdc
ore>javac hello.java
hello.java:38: error: not a statement
tmp68_67;
^
hello.java:40: error: not a statement
int ? = tmp68_67;
^
hello.java:40: error: ';' expected
int ? = tmp68_67;
^
hello.java:40: error: not a statement
int ? = tmp68_67;
^
hello.java:43: error: not a statement
tmp78_74;
^
hello.java:45: error: not a statement
((0x3 ^ 0x5) << 3 ^ 0x2);
^
hello.java:52: error: illegal start of expression
?[tmp102_99] = ((char)(k ^ a.charAt(tmp102_99) ^ n.charAt(j)));
^
hello.java:52: error: illegal start of expression


C:\WINDOWS\system32\cmd.exe
emo\tutorial\step01\files\obf-hello_source_from_Procyon\obf-hello_source_from_Pr
ocyon>javac hello.java
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-D
emo\tutorial\step01\files\obf-hello_source_from_Procyon\obf-hello_source_from_Pr
ocyon>java hello
Obfuscation by Allatori Obfuscator v5.6 DEMO
http://www.allatori.com
Hello World!
C:\Documents and Settings\Administrator\Desktop\Allatori-5.6-Demo\Allatori-5.6-D
emo\tutorial\step01\files\obf-hello_source_from_Procyon\obf-hello_source_from_Pr
ocyon>



With the help of Procyon and JadX, the string obfuscation algorithm that is used by Allatori v5.6 came to light :)



Hope to see you on the next post, have a secure day.

Original Article: Java Kaynak Kodu Dönüştücüleri

Translated to English by: Hüseyin Fatih Akar | Twitter: @thehakar)