Reversing warsaw's Java Crackme 1.1

Introduction

Well, to be frank, this was fun. I downloaded the crackme just for the heck of it & to my surprise, none of the existing regular java decompilers worked. So I thought, what the hell! Let's give it a try. Now, the first thing I did was to open it in my favourite java decompiler jd-gui. As expected, jd-gui couldn't really handle the crackme. Let me attach a screenshot for you.
Nah, its not a complete fail. It does show some interesting stuff but I felt it was unreliable. So, what do you do next? For me, the next obvious step was to put it through some heavy duty tools. **IDA Pro** kinda stands out when you think of heavy duty reversing tools & its not for no reason. It does a pretty good job at that. 😊 So, I just extracted the **Code.class** file from the crackme1.1.jar. Whats left was importing it into IDA & analyzing it. Let me attach another screenshot for you to see how it looked.

The first thing that you see is IDA complaining that something was not right when it tried to disassemble the class file. No surprise there ;) Lets look further & see whats there.
And the whining continues. There is that line of code right at start that’s flagged red by IDA. Before we start burning precious grey cells, let’s analyze further the IDA’s output.

That’s a lot of junk in there. And quite possibly the reason why jd-gui failed to handle it. If am not wrong, its called as ‘junk-byte-injection’ in today’s parlance. And its funny how most of the existing tools fail at handling something as simple as this. So, the next big question is.. what now? I kinda like to be sure that am on the right track before I proceed further. So, just for the heck of it, let’s try to fix the class file.

To do that, first of all let’s see what the opcodes are for this “getstatic 37” instruction.
As it turns out, its “B2 00 25”. Hmm. Another important thing to note here is that this is the only junk instruction in whole of the disassembly produced by IDA. So if we simply replaced this instruction with the equivalent java bytecode's NOP opcode, theoretically the class file should be somewhat fixed. All I did was fire-up my favourite hex-editor & use its crazy ‘Replace All’ option. That calls for another screenshot ;)

Well, yea… that was quick. NOP in Java's Bytecode is 0x00. That's why I simply replaced those 3 bytes with 0x00. It replaced the whole junk in a jiffy . Now its time to see if what we hoped for really worked or we just killed the class file :D. Let's try opening it in jd-gui & see what happens.
Hey! That’s way better! So far, it took us just 20 minutes or so and we already have the code although somewhat broken. But this is good. Am sure the core logic of it is perfect. Now, if we look at the code, it simply takes the argument we pass, multiplies it with \((-37)\), adds 42 to it & then compares it with the magic value of “1720653869” But! Ther’s a catch. 😊 It converts your input to Integer before it multiplies it with \((-37)\). You’ll say, so what? Well, brotha its not that easy of an equation. If you simply look at it from mathematics point of view, the equation turns out to be:

\[
\text{Value} \times (-37) + 42 = 1720653869
\]

So, Value = \((-1720653869-42)/37\).
Hence, Value = \(-46504157.4864865\)

So common-sense dictates that if we pass \(-46504157.4864865\) as the value, we should get the sweet “Correct” message & be done. But nopes, the code won’t take a floating point value. Even if you somehow by your ninja-skillz do manage to pass it, it’ll convert it to type Integer. Ah well! How can I possibly know that for sure? :D I havn’t tried running the app yet. Dumb. Lets check it out.

```java
import java.io.PrintStream;

public final class Code extends PrintStream {

    public static final synchronized void main(String[] paramArrayOfString)
    {
        try
        {
            int &;
            paramArrayOfString = returnAddress;
            System.out.println(Integer.decode(paramArrayOfString[0]), intValue() * \(-37\) + 42 == 1720653869); "Correct" : "Incorrect";
            return;
        }
        finally
        {
            if (true)
               试点工作 = "Please enter a 32bit signed int":
            }
        }
    }
}
```
So that proves it beyond doubt that I was right. Some of the things we can deduce from my futile attempts at trying out the crackme are:

- It won’t take numbers with a decimal in it. (Float values)
- The code we calculated fails miserably even if we try to round it off to the nearest integers.

This is bad. But we are the Ninja-Reversers aren’t we? So well, either we can find out why this happens by googling things related to “use of prime numbers in calculating hash values”, “Integer overflows” & lots more or simply do it the easy-peasy way. Let’s write another java code that reverses the algo & since its pretty simple, bruteforces it to get us the right value. What say? Let me show you the code I made.

```
public class solution {
    public static void main(String[] args) {
        int x=48564157;
        while (true) {
            if((-1)+x+(-37)+42 == 1720653069) {
                System.out.println("Solved! Correct value = "+x+"\n");break;
            }
            x++;
        }
    }
```

Guess what? It works! 😊 Apparently “-975145735” is supposed to be the right value. Let’s give it a try in the actual crackme. Shall we? ;)

```
Solved! Correct value = -975145735
```

And that’s how its done, dutch. I hope you learnt something new out of this tutorial. I sure did enjoy reversing this one. Have a wonderful day ahead!

© Aodrulez

( atul.alex@orchidseven.com )