**XCTF Binary Challenge**

Below is a report on some possible techniques that could be used to solve the binary challenge.

**FLAG 1**

1. Based on file type check, “Main.class” is a data file and “XCTF.class” is a true Java class file.

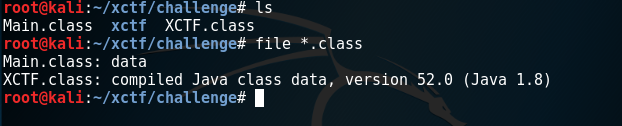


Figure 1 File type check

1. XCTF.class was analysed to be a custom class loader. Insider, there is an findClass which attempts to load class file and pass its content for decryption. The “crypt” function that warrants a deeper analysis to identify if it is possible to decrypt the encrypted class files without using the loader class.

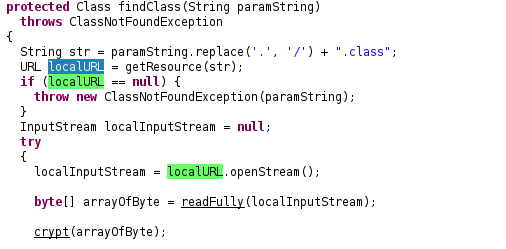


Figure 2 Class Finder Function

1. The decryption (encryption) algorithm appears to be RC4 which contains the standard key scheduling function. RC4 Key initialization with key “nomorepptplease!”. Since the key has 16 characters and we could use openssl to decrypt. Note: Openssl supports 16 bytes and 5 bytes keys even though RC4 is a stream cipher that could support keys of any length.

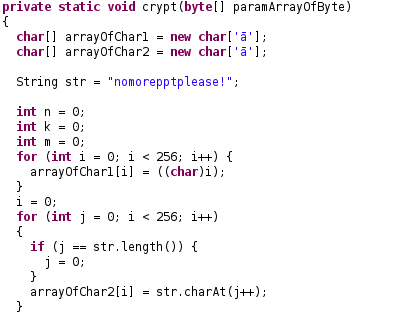


Figure 3 Partial RC 4 Implementation

1. Test decrypt Main.class using the identified RC4 key to confirm that it is truly RC4.

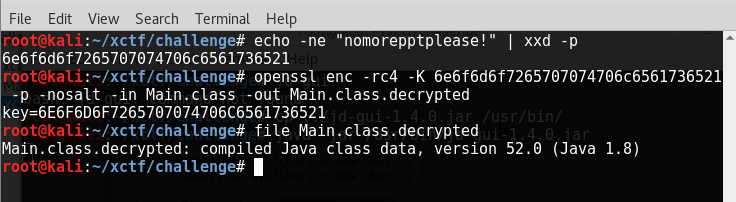


Figure 4 File type check

1. First flag is revealed when “Main.class” is decompiled. It was observed that the flag was passed on to NothingImportant.march\_on as a parameter.

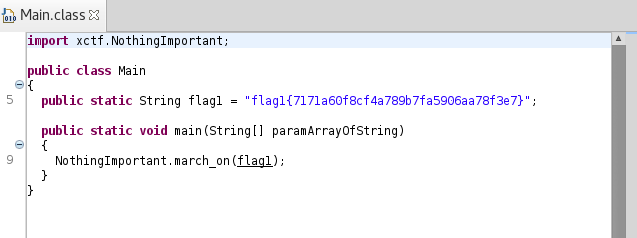


Figure 5 Flag 1 in Main.class

**FLAG 2**

1. Similarly, march\_on was decrypted using the same method for inspection. A number of hints were identified.

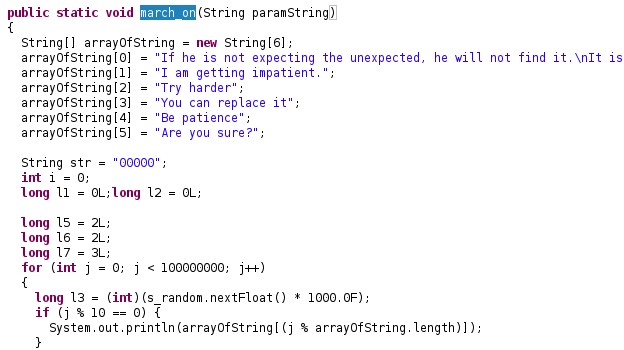


Figure 6 Analysis of march\_on Function

1. It was observed that there is a lot of time manipulations which causes the delay. Hence, we would need to think of ways to bypass the sleep. One way is to analyse how the hash was constructed, and we could potentially reconstruct it offline.
2. The second flag was observed to be a hash based on the following chained hashes.

*FLAG2 = MD5 {*

*MD5{*

*MD5{ …*

*MD5 {*

*MD5 { “flag1{7171a60f8cf4a789b7fa5906aa78f3e7}” + 0 }*

*+ 1 }*

*…}*

*+ (n-1) }*

*+ n }*

*, where n is a smallest possible natural number such that the hex value of FLAG2 is computed to start with “00000”.*

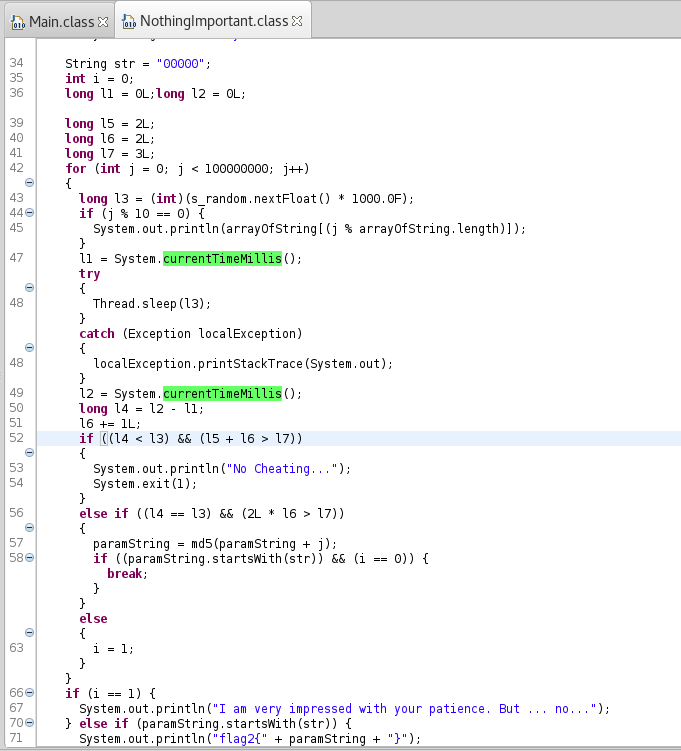


Figure 7 Analysis of Flag2 Generation Function

1. Based on analysis, there are possibly 3 or more ways to get flag 2.

**Method 1 –** Decompile and remove the restriction codes such as sleep and checks. Compile the cleaned code. Risk of identifying the correct part to remove is a challenge.

**Method 2 –** Write a flag generator based on analysis. This requires strong confidence of analysis to confirm the flag is truly what it should be.

**Method 3 –** Assuming too lazy to figure out how to get how flag 2 is generated. We could let the code to generate out the flag on its own. This could be done by “hijacking” the System class and the Thread class. We could possibly create modified System and Thread classes and put them in the same package as “NothingImportant.class” – “xctf”. It is important to update the implied package of “System” and “Thread” classes inside “NothingImportant.class”

**Method 3**

1. Write a fake System class with some accounting to keep track how many milliseconds was sleep as it was observed that NothingImportant.class expect the elapsed sleep time to be accurately tracked, where is likely not going to happen in the real world assuming there are some time clocked for other instructions.

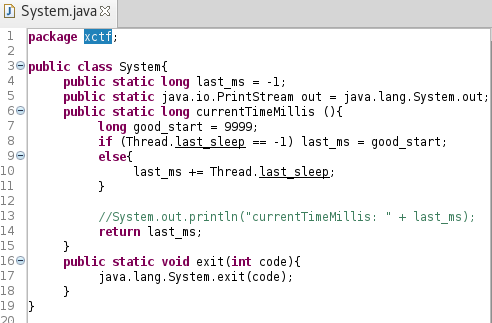


Figure 8 Fake System Class

1. Write a fake Thread class that only tracks sleep time invoked to facilitate use by System class.

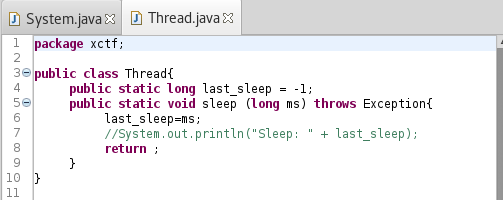


Figure 9 Fake Thread Class

1. Use class editor to update “NothingImportant.class” to use change references from “java.lang.Thread” to “xctf.Thread” and from “java.lang.System” to “xctf.System”.

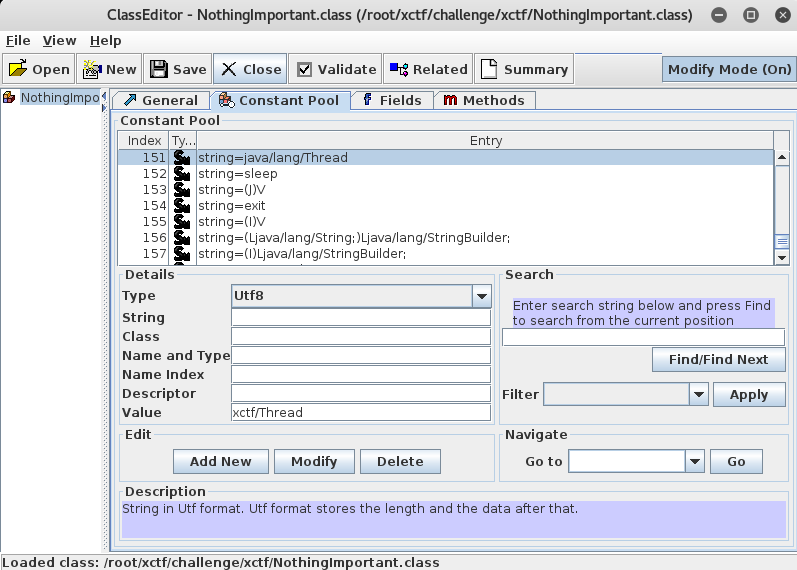


Figure 10 Class Editing to update the referenced class

1. Encrypt all the classes and run the main class – “XCTF.class”. This would accelerate all the random sleep time and speedily reveal that the second flag is “flag2{000006e2cf249106e954a67d8865c7b2}”.

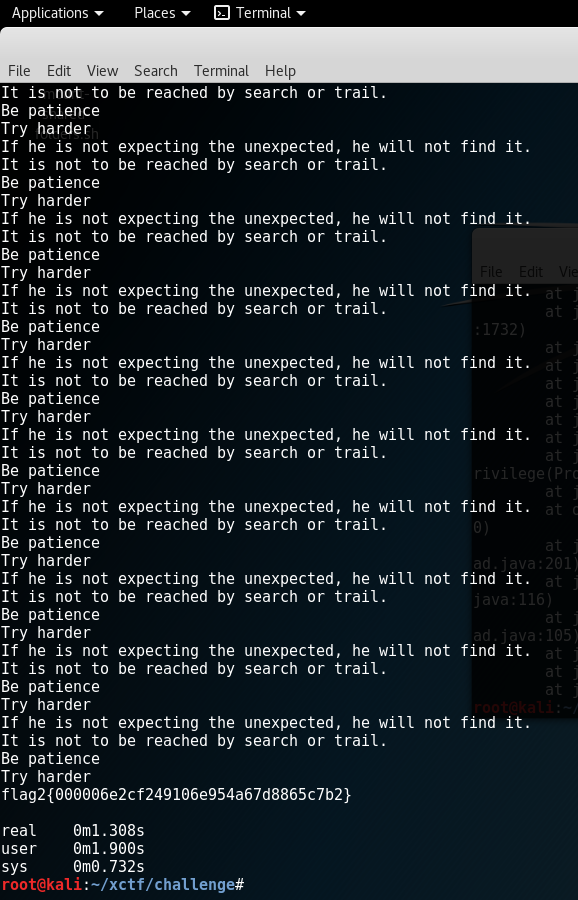


Figure 11 Run with Sleep Accelerated