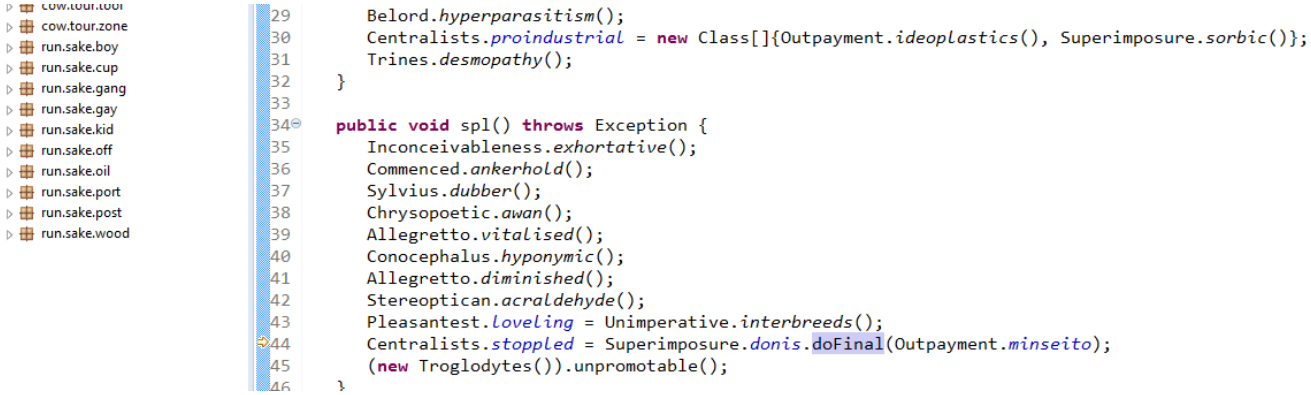
Qealler Infostealer static analysis – Part 0x1

securityinbits.com/malware-analysis/pyrogenic-infostealer-static-analysis-part-0x1/

January 6, 2020



Qealler is heavily obfuscated Java based Infostealer which is quite active based on ANY.RUN submission. This will be a three part blog series, this post will focus on Qealler/Pyrogenic static analysis, next part 0x2 we learn unpacking using Java agent and in the last part 0x3 we find similarity between Qealler/Pyrogenic variants based on static code analysis . You may download the BankPaymAdviceVend_LLCRep.jar from ANY.RUN (MD5: F0E21C7789CD57EEBF8ECDB9FADAB26B) and follow along or download the latest Qealler sample from ANY.RUN submission.

CONTENTS

Overview

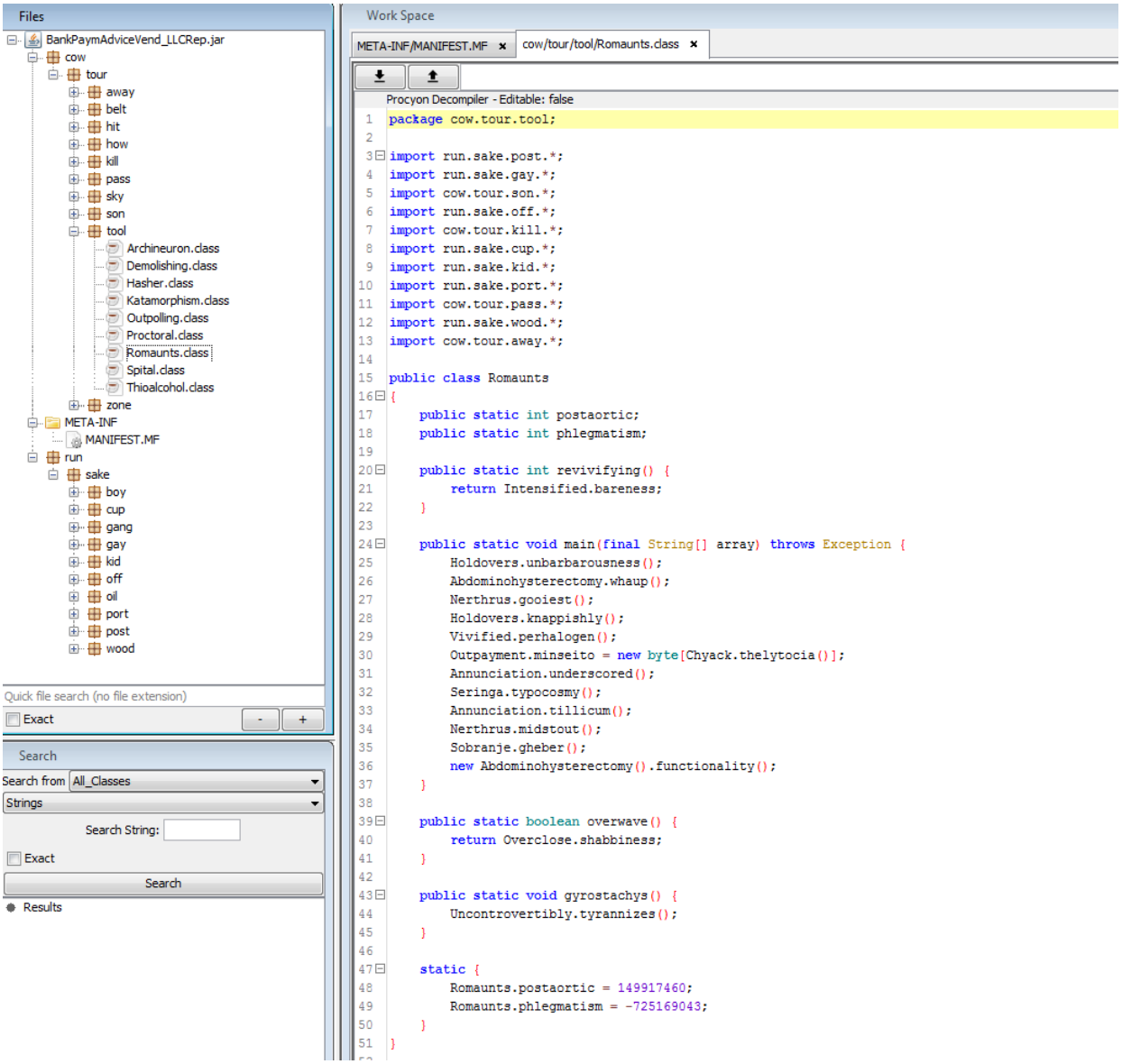
It's currently targeting different regions e.g. Australian companies[1], Africa and the Middle East[2] based on the references. I will be using Bytecode Viewer to decompile Jar using FernFlower Java Decompiler. Let's start with quick dynamic analysis. Our main goal for the blog series is to unpack this jar so we can analyse the capability and compare it with Qealler.

Quick Dynamic Analysis

- Connect to CC 157.245.160[.]150 at port 80 and create the below process. `cmd.exe /c chcp 1252 > NUL & powershell.exe -ExecutionPolicy Bypass -NoExit -NoProfile -Command -`
- Drop these two clean files sqlitejdbc.dll (MD5: a4e510d903f05892d77741c5f4d95b5d) and jnidispatch.dll (MD5: d2f0da769204b8c45c207d8f3d8fc37e) but it deletes these two file before exiting.
- Connect to bot.whatismyipaddress.com to get the public IP of the infected system.
- Steal credential from different applications

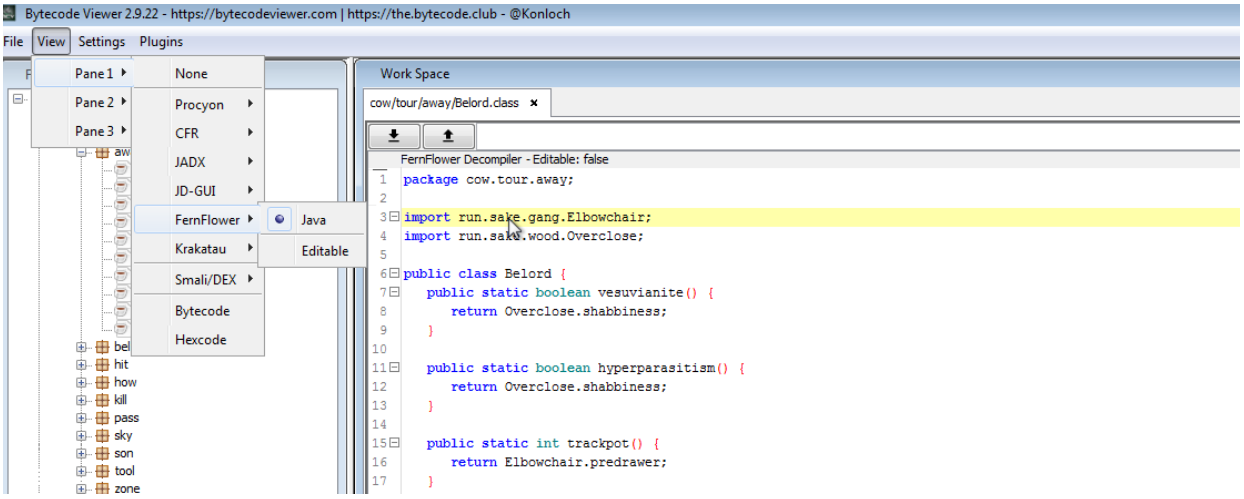
Packed Pyrogenic static analysis

1. Open the jar file in BCV (Bytecode Viewer), you will see multiple class files in different packages. Below pic shows the main entry point of the jar file.



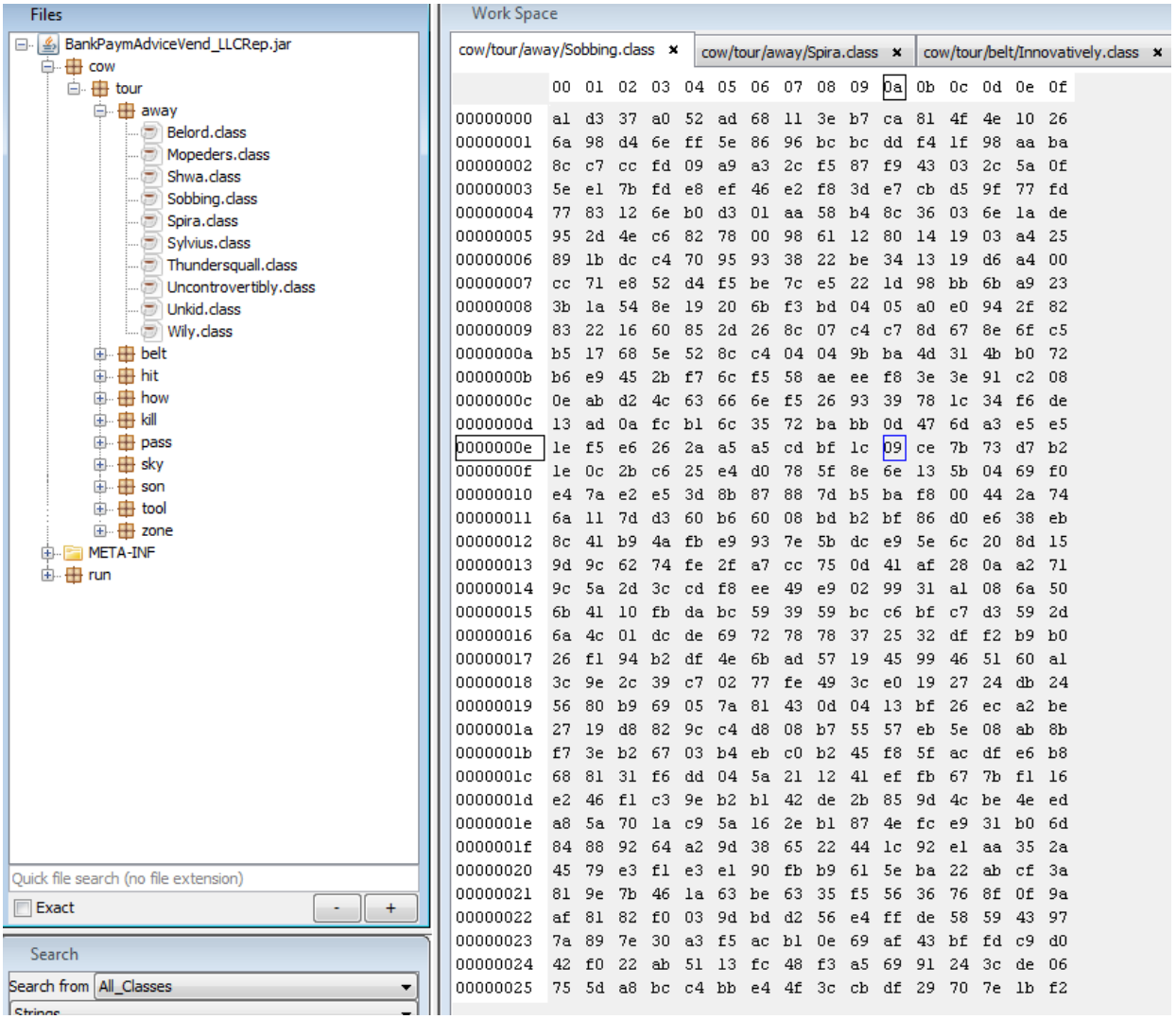
Packed Pyrogenic BCV

2. For this sample, I found out that FernFlower decompiled the source code correctly. Select View -> Pane 1 -> FernFlower -> Java in BCV as shown below.



Bytecode Viewer FernFlower selection

3. If you browse the different class files in BCV, you will find many encrypted class files which don't translate to Java src code. e.g. one of them is shown below



Pyrogenic encrypted class

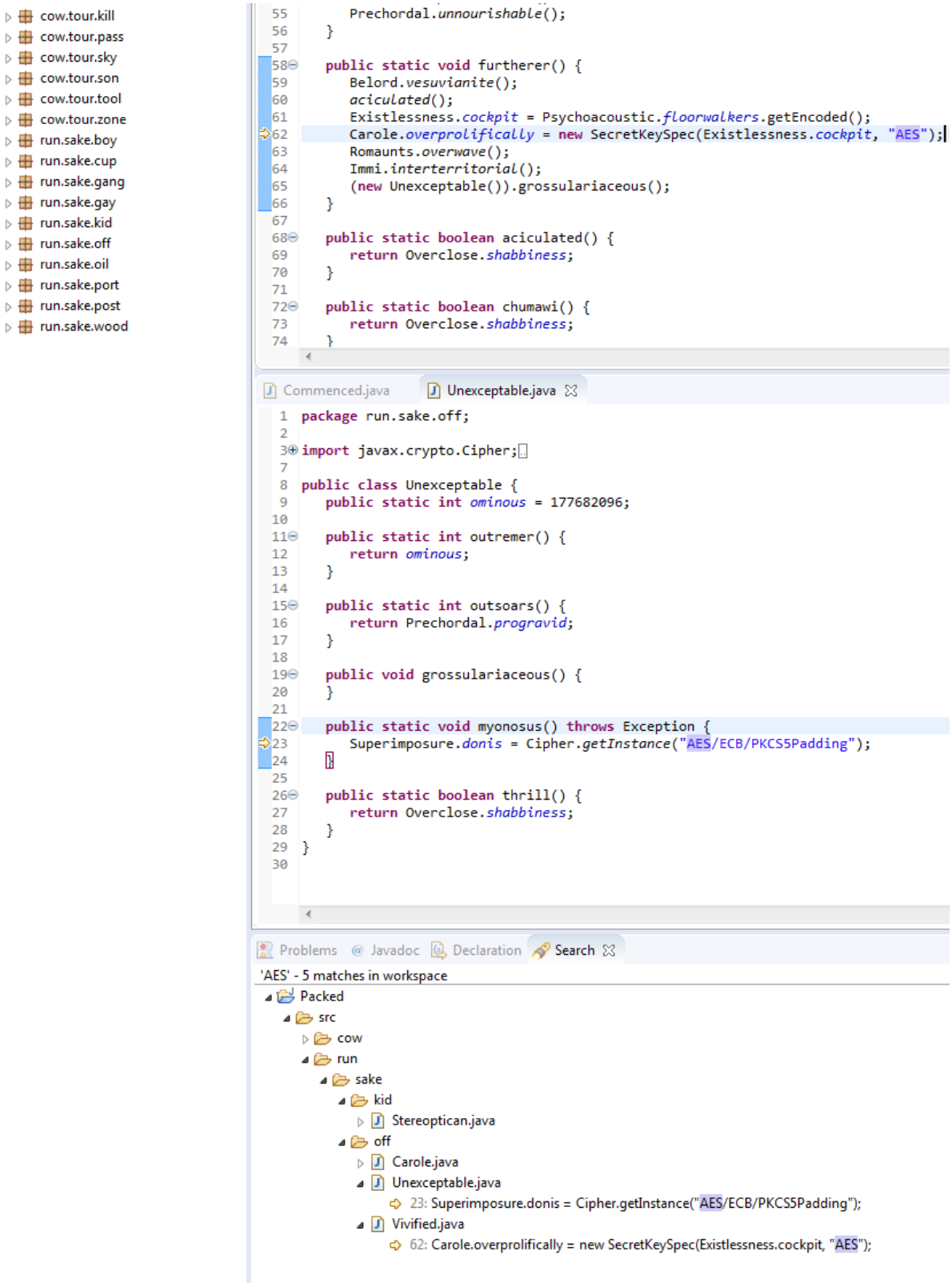
4. Based on the above encrypted class file, you can guess that there should be some decryption algorithm used to decrypt those files.
5. Decryption algorithms can be custom or well known e.g. AES. Study this example java code [3] which encrypt/decrypt using AES.

```java
public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
}
```

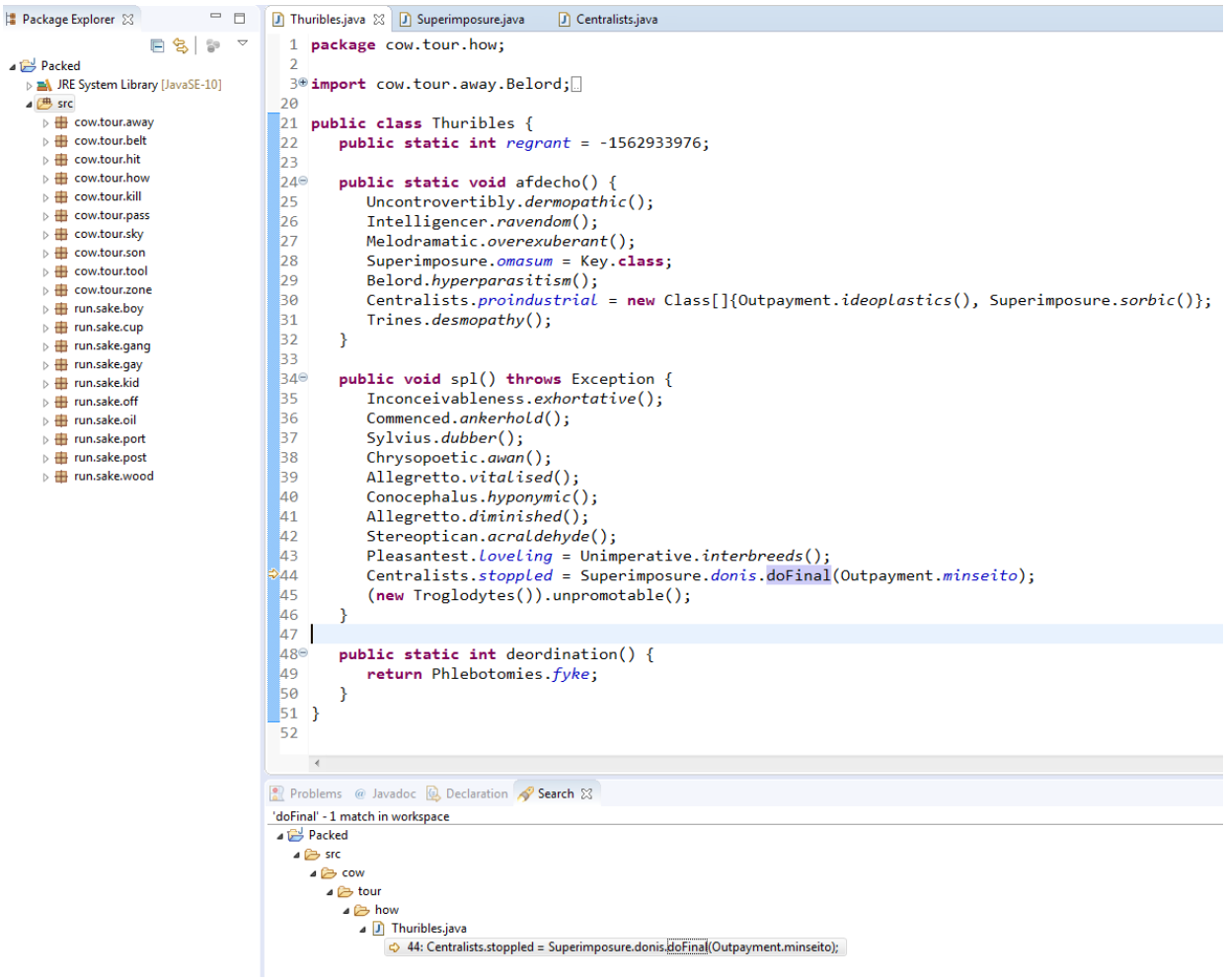
AES Java Encrypt Decrypt Example

Some of the keyword mentioned in the above java code e.g. **getInstance** can help us to find the encryption algorithm and **doFinal** can point to final decryption result.

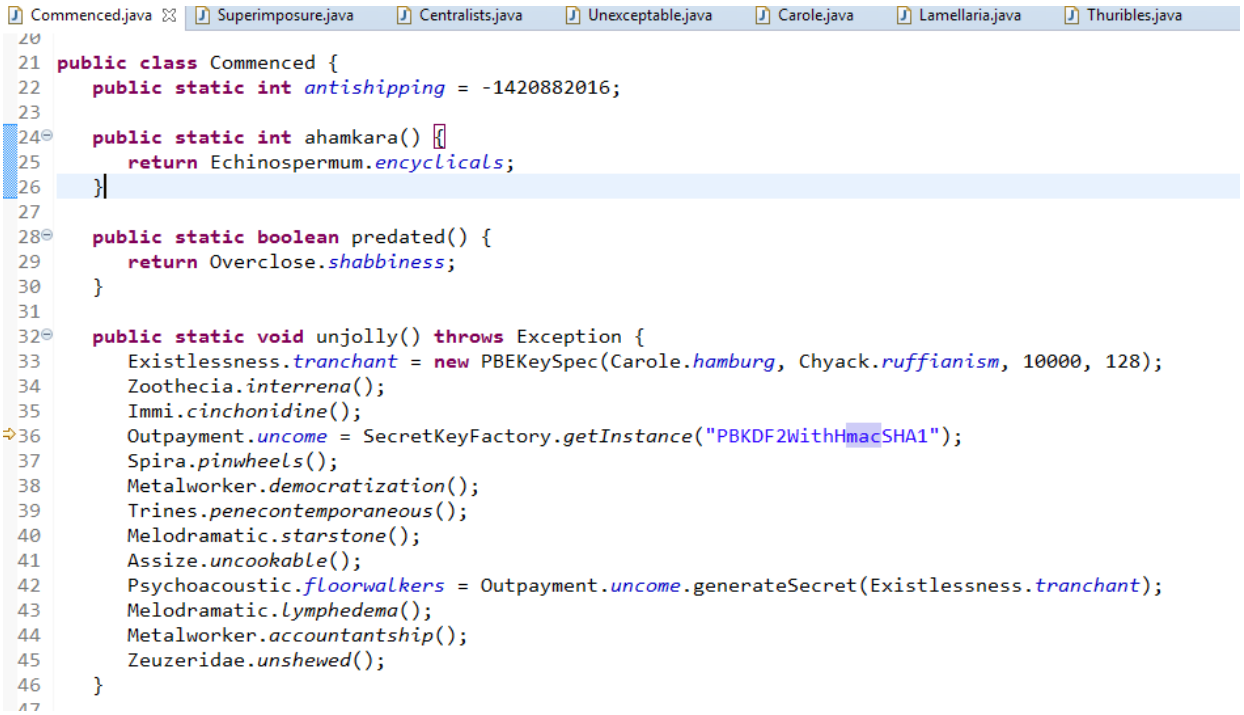
6. Let’s search for AES references after importing the decompiled src code to Eclipse IDE.



Pyrogenic AES references



Pyrogenic doFinal reference



Pyrogenic PBKDF2WithHmacSHA1 reference

7. Based on the above images which shows multiple references, we can confirm that this sample uses the algo **“AES/ECB/PKCS5Padding”** and key may be generated using **“PBKDF2WithHmacSHA1”** . So it confirmed that it doesn’t use any custom decryption algorithm.
8. We can add our code to write the data to file after **doFinal** call and execute the sample in IDE to get the dumped class file Then we can decompile the class file using BCV and continue analysis. But it can be multiple layer obfuscation which can make our analysis harder and slower.

Conclusion

This above static analysis method to find the encryption routine and interesting breakpoint (doFinal) while debugging is very useful in Java Malware analysis. Using this approach you will not miss any code path but this requires more time and effort. So in the upcoming part 0x2 , we will unpack this malware using **Java agent** which will speed up our analysis.

Hope you enjoyed this post, please Follow @Securityinbits **me** on Twitter to get the latest update about my malware analysis & DFIR journey. Happy Reversing 😊