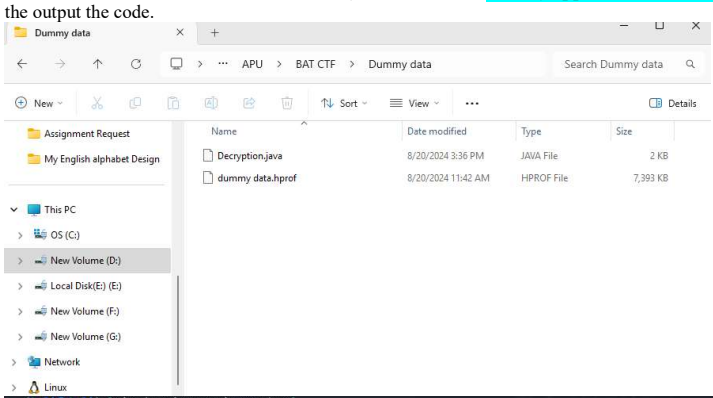
BAT--Dummy data

Monday, March 10, 2025 11:43 AM

The APU servers have been **compromised妥协（遭到攻击）**, resulting in all files being locked with an unknown encryption method. One of our students Wala was tasked with investigating the attack to find a way to recover the critical data. During the investigation, Wala discovered that while most files were encrypted, one Java code file remained **intact完好无损**. Additionally, Wala managed to capture memory dump data, leading to the discovery of a mysterious file that couldn't be opened. At this point, your mission is to **delve深入** deeper into the investigation. Your objectives are to understand the significance of the unencrypted Java file and to uncover the contents of the mysterious file that **initially appeared inaccessible最初似乎无法进入** and find out the output the code.




```
┌──(kali㉿kali)-[~/Desktop/CTF BAT/Dummy data]
└─$ cat Decryption.java
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.spec.IvParameterSpec;

public class Decryption {

    public static void main(String[] args) throws InterruptedException {
        String encryptedSecret = "fc34fa4110356538528a3e29e60750d95135fd78919550401e38afa407bb5a8349a869454f85676afb54c18a106ae003b1c6cb2e8207de4e081b80a0b7b0be
e21da1100a026112b377888744a058cceaa6d6f1d4e1407c21e9d3c7e878c0631b4";

        try {
            String decryptedPassword = decrypt(encryptedSecret, Key, ivHex);
            System.out.println("..................................................................................................................
..................................................." + decryptedPassword);
        } catch (Exception e) {
            System.out.println("Decryption failed: " + e.getMessage());
        }

        Thread.sleep(5000);
    }

    public static String decrypt(String encryptedHex, String hexKey, String ivHex) throws Exception {
        byte[] keyBytes = hexStringToByteArray(hexKey);
        byte[] ivBytes = hexStringToByteArray(ivHex);

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
        cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
        byte[] encryptedBytes = hexStringToByteArray(encryptedHex);

        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

        return new String(decryptedBytes);
    }

    public static byte[] hexStringToByteArray(String s) {
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                    + Character.digit(s.charAt(i + 1), 16));
        }
        return data;
    }
}
```

The Main Method

```
public static void main(String[] args) throws
InterruptedException {
    String encryptedSecret =
"fc3f4a4110356538528a3e29e60750d95135fd789195
50401e38afa407b...";
try {
        String decryptedPassword =
decrypt(encryptedSecret, Key, ivHex);

        // Output the results

System.out.println("..................................................
.....................................................");
        System.out.println("Decrypted Password: " +
decryptedPassword);
    } catch (Exception e) {
        System.out.println("Decryption failed: " +
e.getMessage());
    }
Thread.sleep(5000);
}
```

decrypt Method

```
public static String decrypt(String encryptedHex,
String hexKey, String ivHex) throws Exception {
    byte[] keyBytes =
hexStringToByteArray(hexKey);
    byte[] ivBytes = hexStringToByteArray(ivHex);
Cipher cipher =
Cipher.getInstance("AES/CBC/PKCS5Padding");
    SecretKeySpec keySpec = new
SecretKeySpec(keyBytes, "AES");
    IvParameterSpec ivSpec = new
IvParameterSpec(ivBytes);
    cipher.init(Cipher.DECRYPT_MODE, keySpec,
ivSpec);
byte[] encryptedBytes =
hexStringToByteArray(encryptedHex);
    byte[] decryptedBytes =
cipher.doFinal(encryptedBytes);
return new String(decryptedBytes);
}
```

hexStringToByteArray Method

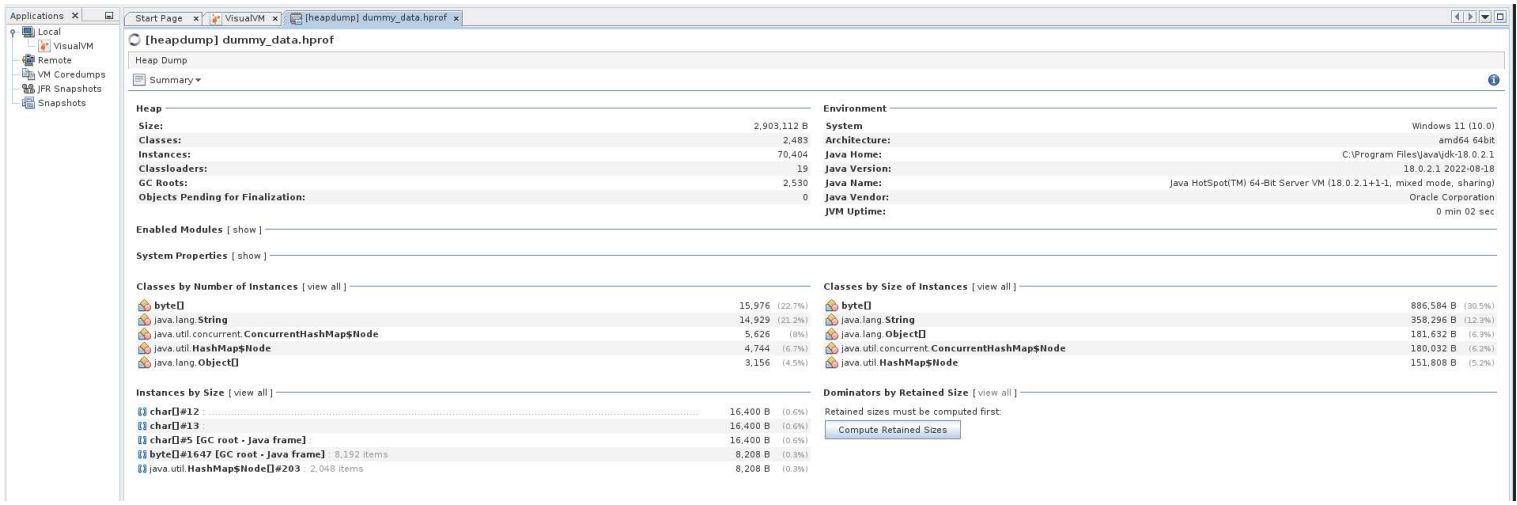
```
public static byte[] hexStringToByteArray(String s) {
    int len = s.length();
    byte[] data = new byte[len / 2];
    for (int i = 0; i < len; i += 2) {
        // Convert each pair of hex characters to a byte
        data[i / 2] = (byte) ((Character.digit(s.charAt(i),
16) << 4) +
                              Character.digit(s.charAt(i + 1),
16));
    }
    return data;
}
```

How to Open a .hprof File:

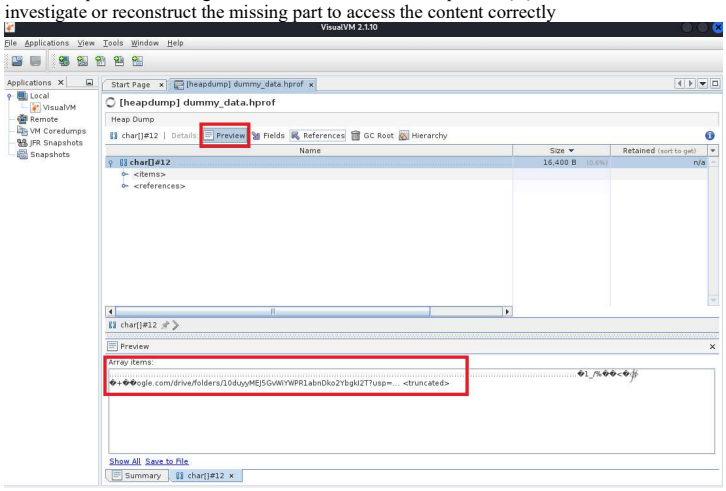
You can open and analyze a .hprof file using **VisualVM**, a tool that provides a visual interface for monitoring and troubleshooting Java applications. After downloading and installing VisualVM, you can load the .hprof file to explore the heap dump and identify any potential issues with your Java application. By analyzing the .hprof file, it may be possible to discover sensitive information such as the Initialization Vector (IV) and the private key, if they were stored in memory at the time the heap dump was taken. This can be crucial for decrypting data encrypted with the AES algorithm, as both the IV and the private key are necessary for successful decryption.

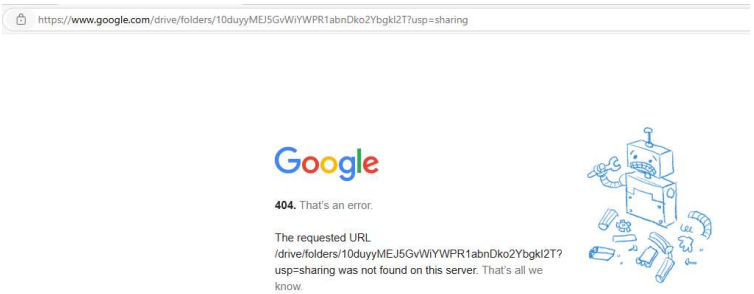
VisualVM Fouction

•Heap Size: Shows the total memory size used.
•**Instances实例**: Number of objects in memory.
•Classes: Number of unique classes.
•Environment: Details like system architecture, Java version, etc.
•Classes by Number of Instances: Lists the classes with the most instances, like byte[] and String.



In the “Instances by Size” section of VisualVM, the entry char[]#12 appears, which could be significant. This aligns with the earlier analysis of the code where the flag is expected to be printed after a sequence of dots (.....). This char[]#12 instance might contain the actual flag or related data, potentially hidden after these dots. By examining this specific instance, you may uncover the flag or sensitive information within like:
This URL points to a Google Drive folder, but the middle portion (...) seems to be obfuscated or incomplete. You might need to investigate or reconstruct the missing part to access the content correctly





We still haven’t found what we are looking for, to locate the IV and KEY, you can navigate to the Decryption.main class under the threats category. By examining the local variables in this class, you can preview and retrieve the IV, KEY, and the encrypted string needed for decryption. This will provide the necessary information to successfully decrypt the string using the AES algorithm.