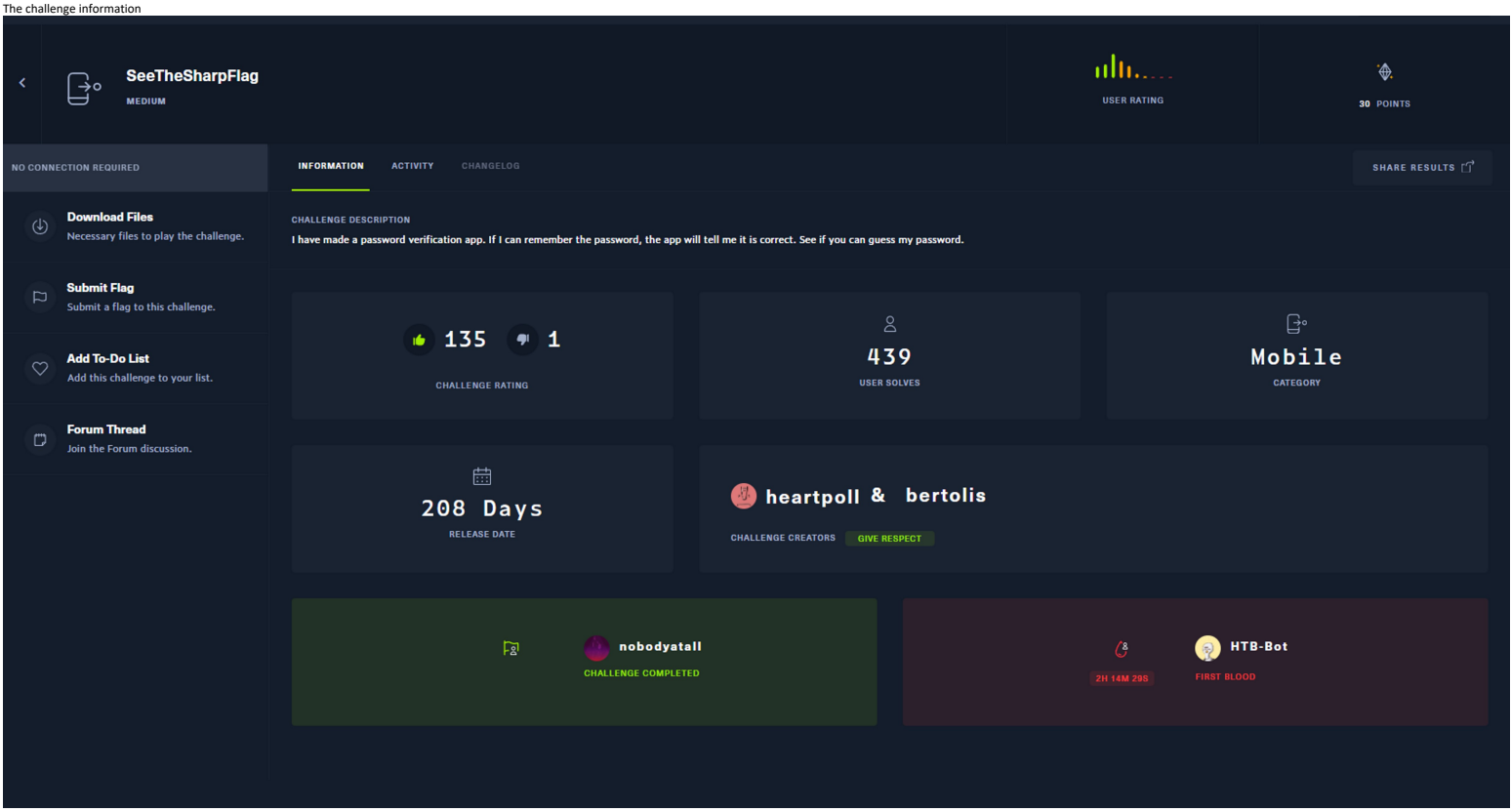
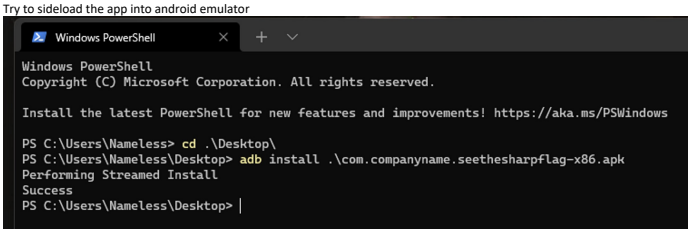
Challenge Walkthrough

Thursday, March 3, 2022 7:53 PM

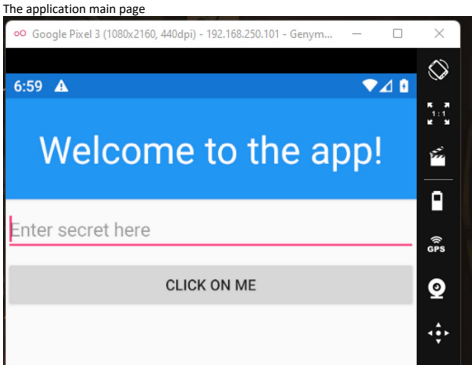
The challenge information



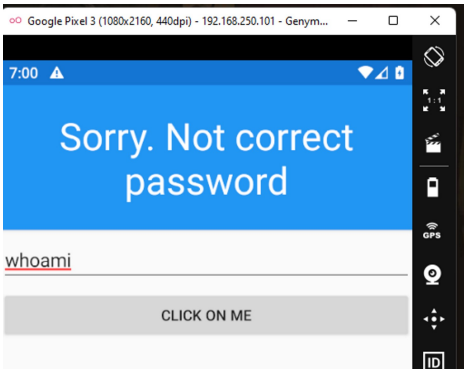
Try to sideload the app into android emulator



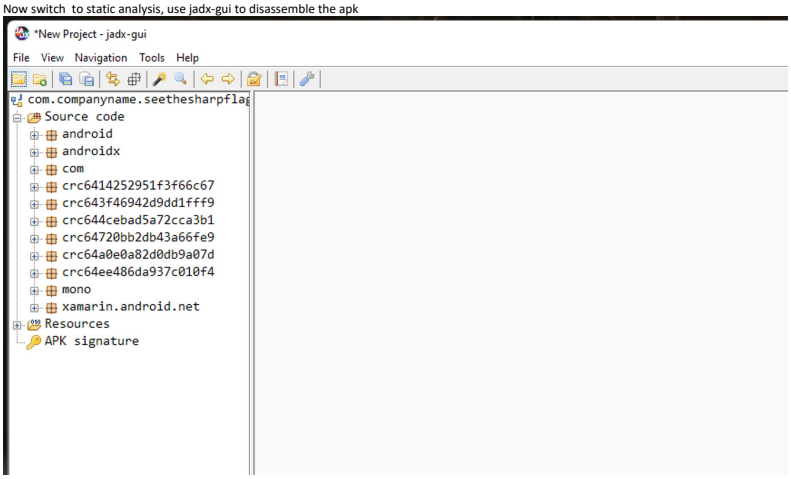
The application main page



Type in random text & it shows wrong password



Now switch to static analysis, use jadx-gui to disassemble the apk



Now 1st step, check out the AndroidManifest.xml

Over here we can see that this are the package & classname of the Main Activity for this apk

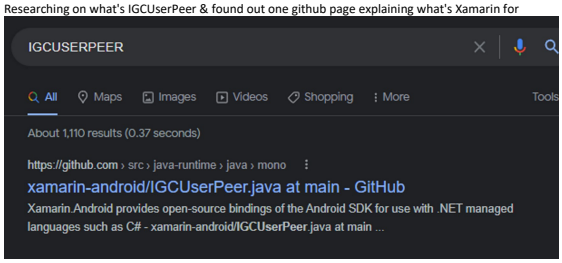
```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<application android:theme="@style/MainTheme" android:label="SeeTheSharpFlag.Android" android:icon="@mipmap/icon" android:name="android.app.Application" android:allowBackup="true" android:extractNativeLibs="true"
    <activity android:theme="@style/MainTheme" android:label="SeeTheSharpFlag" android:icon="@mipmap/icon" android:name="crc644cebad5a72cca3b1.MainActivity" android:configChanges="orientation|screenLayout|uiMode|
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <receiver android:name="crc643f46942d9dd1fff9.PowerSaveModeBroadcastReceiver" android:enabled="true" android:exported="false"/>
    <provider android:name="mono.MonoRuntimeProvider" android:exported="false" android:authorities="com.companyname.seethesharpflag.mono.MonoRuntimeProvider.__mono_init__" android:initOrder="1999999999"/>
</application>
```

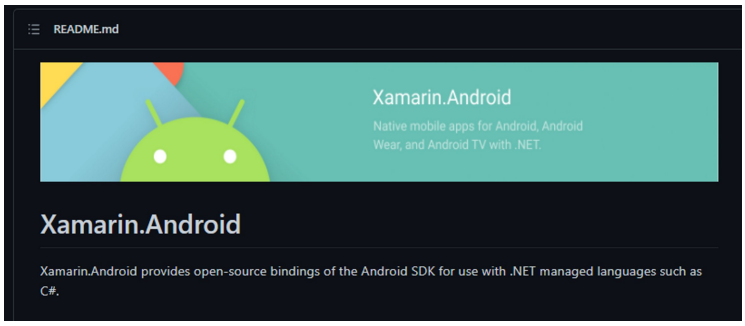
Over here we can see that the class implements IGCUserPeer

AndroidManifest.xml | crc644cebad5a72cca3b1.MainActivity

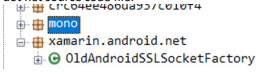
```java
package crc644cebad5a72cca3b1;

import android.os.Bundle;
import crc643f46942d9dd1fff9.FormsAppCompatActivity;
import java.util.ArrayList;
import mono.android.IGCUserPeer;
import mono.android.Runtime;
import mono.android.TypeManager;

public class MainActivity extends FormsAppCompatActivity implements IGCUserPeer {
    public static final String __md_methods = "n_onCreate:(Landroid/os/Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\nn_onRe
    private ArrayList refList;

    private native void n_onCreate(Bundle bundle);

    private native void n_onRequestPermissionsResult(int i, String[] strArr, int[] iArr);

    static {
        Runtime.register("SeeTheSharpFlag.Droid.MainActivity, SeeTheSharpFlag.Android", MainActivity.class, __md_methods);
    }

    public MainActivity() {
```

Researching on what's IGCUserPeer & found out one github page explaining what's Xamarin for





& noticed that in the apk there's Xamarin library too, so most likely all the real MainActivity codes are in dot net source code file.



Based on the source code here, we noticed that it tried to register & activate something from this 2 files

```java
static {
    Runtime.register("SeeTheSharpFlag.Droid.MainActivity, SeeTheSharpFlag.Android", MainActivity.class, __md_methods);
}

public MainActivity() {
    if (getClass() == MainActivity.class) {
        TypeManager.Activate("SeeTheSharpFlag.Droid.MainActivity, SeeTheSharpFlag.Android", "", this, new Object[0]);
    }
}

public MainActivity(int i) {
    super(i);
    if (getClass() == MainActivity.class) {
        TypeManager.Activate("SeeTheSharpFlag.Droid.MainActivity, SeeTheSharpFlag.Android", "System.Int32, mscorlib", this, new Object[]{Integer.valueOf(i)});
    }
}
```

Now use apktool to decompress the apk

```
┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$ apktool d com.companyname.seethesharpflag-x86.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.5.0-dirty on com.companyname.seethesharpflag-x86.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Search for anything that related to the one the source code tried to read & found out there's 2 dll file inside the assemblies directory

```
┌──(kali㉿0×DEADBEEF)-[~/Desktop/com.companyname.seethesharpflag-x86]
└─$ find . -name 'SeeTheSharp*' -type f 2>/dev/null
./unknown/assemblies/SeeTheSharpFlag.Android.dll
./unknown/assemblies/SeeTheSharpFlag.dll
```

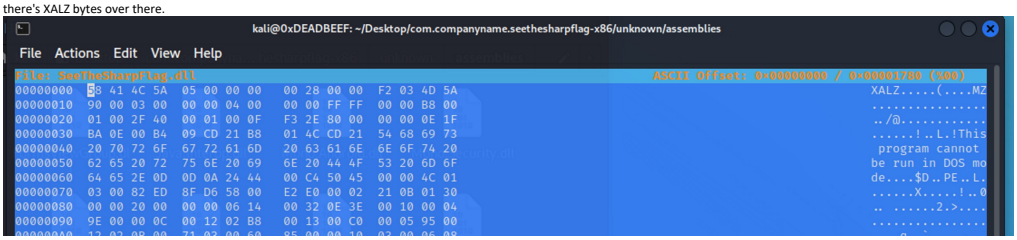
In the directory, noticed that there's bunch of dll files and some of them are xamarin dll.

```
┌──(kali㉿0×DEADBEEF)-[~/Desktop/com.companyname.seethesharpflag-x86/unknown/assemblies]
└─$ ls
FormsViewGroup.dll                 System.Xml.dll                                  Xamarin.AndroidX.Loader.dll
Java.Interop.dll                   Xamarin.AndroidX.Activity.dll                   Xamarin.AndroidX.RecyclerView.dll
Mono.Android.dll                   Xamarin.AndroidX.AppCompat.AppCompatResources.dll  Xamarin.AndroidX.SavedState.dll
Mono.Security.dll                  Xamarin.AndroidX.AppCompat.dll                  Xamarin.AndroidX.SwipeRefreshLayout.dll
mscorlib.dll                       Xamarin.AndroidX.CardView.dll                   Xamarin.AndroidX.ViewPager.dll
SeeTheSharpFlag.Android.dll        Xamarin.AndroidX.CoordinatorLayout.dll          Xamarin.Essentials.dll
SeeTheSharpFlag.dll                Xamarin.AndroidX.Core.dll                       Xamarin.Forms.Core.dll
System.Core.dll                    Xamarin.AndroidX.CustomView.dll                 Xamarin.Forms.Platform.Android.dll
System.dll                         Xamarin.AndroidX.DrawerLayout.dll               Xamarin.Forms.Platform.dll
System.Drawing.Common.dll          Xamarin.AndroidX.Fragment.dll                   Xamarin.Forms.Xaml.dll
System.Net.Http.dll                Xamarin.AndroidX.Legacy.Support.Core.UI.dll     Xamarin.Google.Android.Material.dll
System.Numerics.dll                Xamarin.AndroidX.Lifecycle.Common.dll           Xamarin.Google.Guava.ListenableFuture.dll
System.Runtime.Serialization.dll   Xamarin.AndroidX.Lifecycle.LiveData.Core.dll
System.ServiceModel.Internals.dll  Xamarin.AndroidX.Lifecycle.ViewModel.dll
```

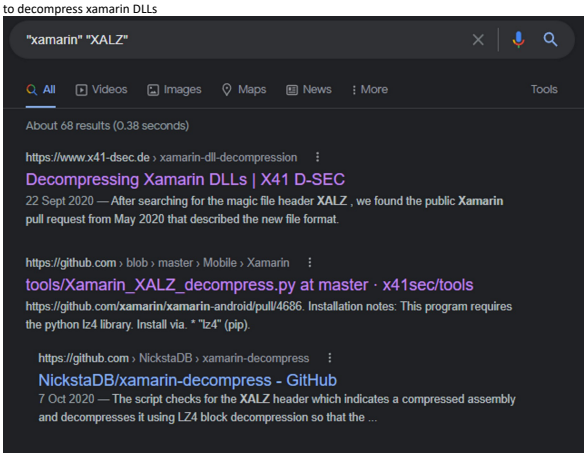
Noticed that the dll here are detected as Sony PlayStation Audio magic header but normal dll file should be PE exe or a linker file magic header

```
┌──(kali㉿0×DEADBEEF)-[~/Desktop/com.companyname.seethesharpflag-x86/unknown/assemblies]
└─$ file SeeTheSharpFlag.dll
SeeTheSharpFlag.dll: Sony PlayStation Audio
```

Now try to read the dll file in hexeditor & found out that this program had PE exe header but before that there's XALZ bytes over there.



Now try to do some research regarding to Xamarin & XALZ header & found out that there's some ways to decompress xamarin DLLs



Let's try to understand how the whole program structure works
// Link: https://www.x41-dsec.de/security/news/working/research/2020/09/22/xamarin-dll-decompression/

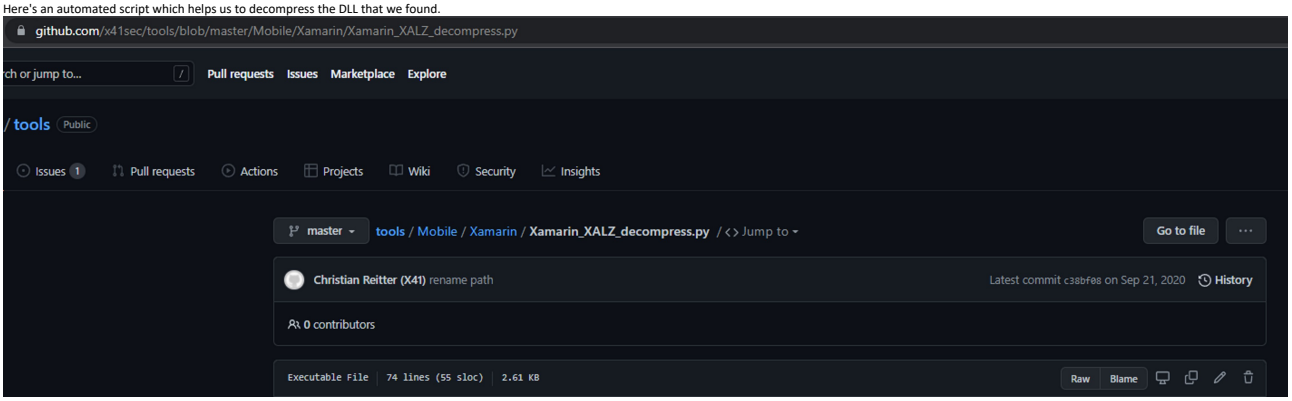
Building a Solution

Through the public documentation, we knew the file format:

```
[ 4 byte magic header ]
[ 4 byte header index ]
[ 4 byte uncompressed payload length ]
[ rest: lz4 compressed payload ]
```

Using this, we were able to whip up a quick python script that could decompress the DLLs.

Here's an automated script which helps us to decompress the DLL that we found.



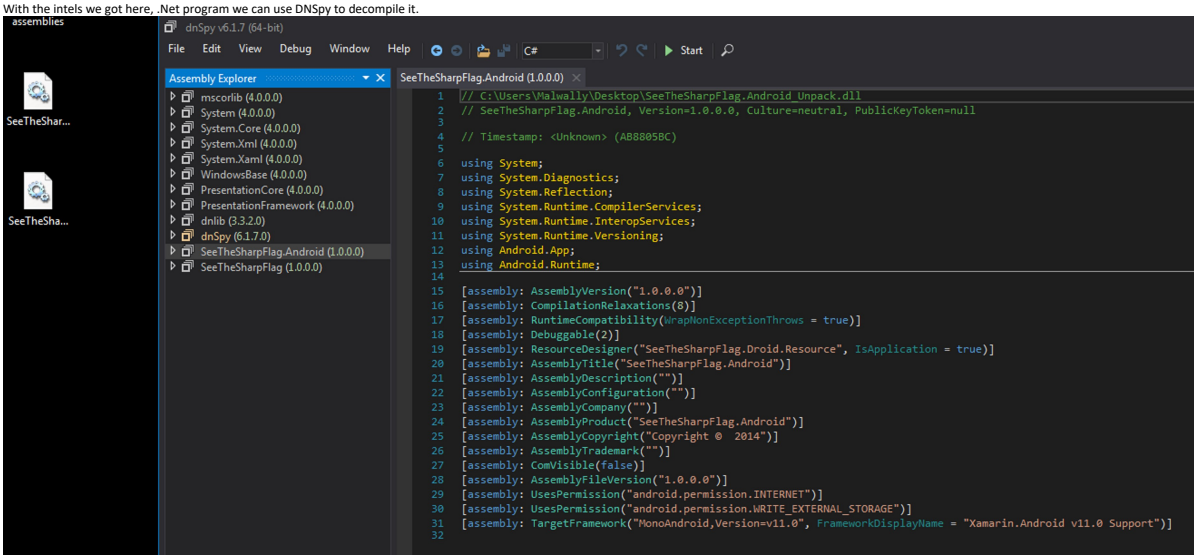
Now try to decompress the DLL file with Xamarin_XALZ_decompress.py

```
File  Actions  Edit  View  Help

┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$ python3 Xamarin_XALZ_decompress.py com.companyname.seethesharpflag-x86/unknown/assemblies/SeeTheSharpFlag.Android.dll SeeTheSharp
Flag.Android_Unpack.dll
header index: b'\x04\x00\x00\x00'
compressed payload size: 156652 bytes
uncompressed length according to header: 302080 bytes
result written to file

┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$ python3 Xamarin_XALZ_decompress.py com.companyname.seethesharpflag-x86/unknown/assemblies/SeeTheSharpFlag.dll SeeTheSharpFlag_Unp
ack.dll
header index: b'\x05\x00\x00\x00'
compressed payload size: 6005 bytes
uncompressed length according to header: 10240 bytes
result written to file

┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$
```

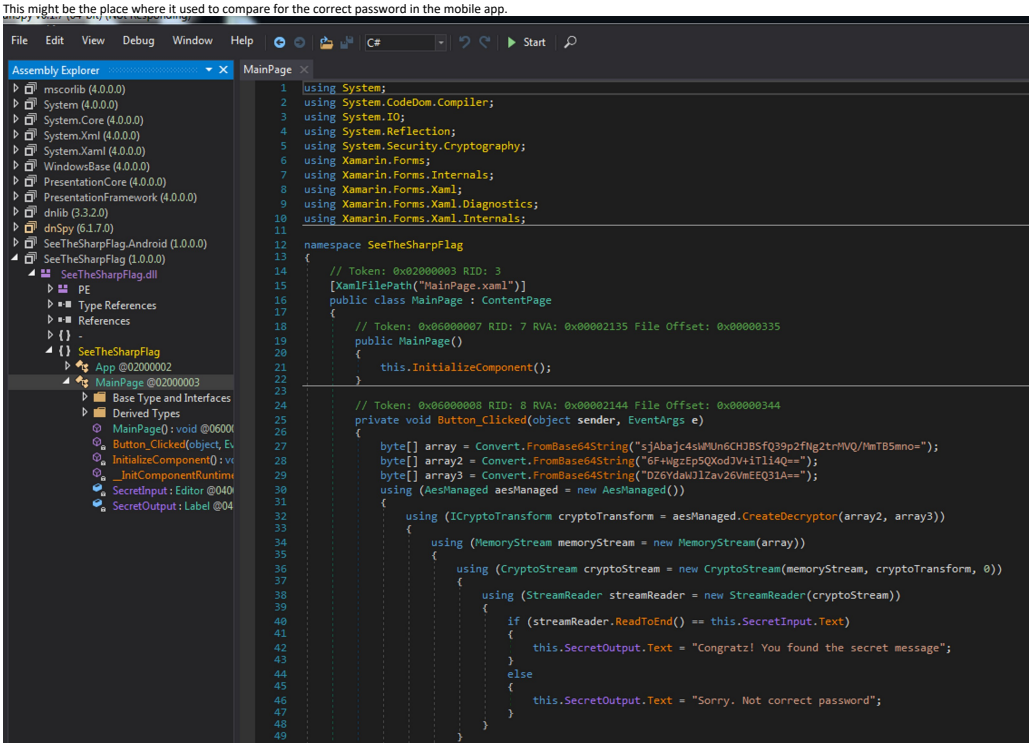
If we check using file command again & now we noticed that it's PE EXE format + it's .Net assembly here

```
┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$ file SeeTheSharpFlag.Android_Unpack.dll
SeeTheSharpFlag.Android_Unpack.dll: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows

┌──(kali㉿0×DEADBEEF)-[~/Desktop]
└─$
```

With the intels we got here, .Net program we can use DNSpy to decompile it.



In the SeeTheSharpFlag.dll we noticed that this part of MainPage class code there's some text which print out some text on the screen.
This might be the place where it used to compare for the correct password in the mobile app.



The encryption that this password used are AES

```
using (AesManaged aesManaged = new AesManaged())
{
```

https://docs.microsoft.com › en-us › dotnet › api › syste...

AesManaged Class (System.Security.Cryptography)

The AES algorithm is essentially the Rijndael symmetric algorithm with a fixed block size and iteration count. This class functions the same way as the ...

This part we can see there's 3 array over here in base64 encoded string

```
private void button_clicked(object sender, EventArgs e)
{
    byte[] array = Convert.FromBase64String("sjAbajc4sWMUn6CHJBSfQ39p2fNg2trMVQ/MmTB5mno=");
    byte[] array2 = Convert.FromBase64String("6F+WgzEp5QXodJV+iTli4Q==");
    byte[] array3 = Convert.FromBase64String("DZ6YdaWJlZav26VmEEQ31A==");
    using (AesManaged aesManaged = new AesManaged())
```

The first part it used aesManaged.CreateDecryptor() with array2 & array 3 as the param

```
using (AesManaged aesManaged = new AesManaged())
{
    using (ICryptoTransform cryptoTransform = aesManaged.CreateDecryptor(array2, array3))
    {
```

While researching on what this method works, we found a microsoft doc explaining the method usage

Here we noticed that the 1st param are the key, & the 2nd param are the IV

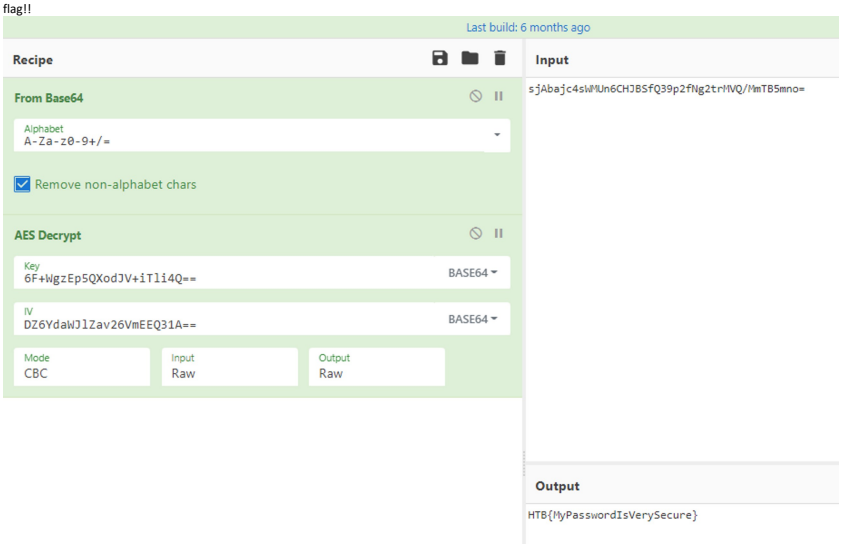
CreateDecryptor(Byte[], Byte[])	Creates a symmetric decryptor object using the specified key and initialization vector (IV).

Now from this part, we got the idea that
Array 2 => Key
Array 3 => IV

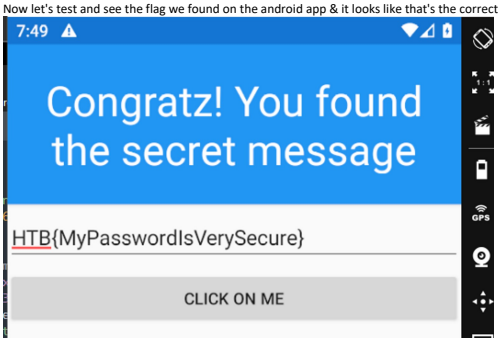
Now let's check out what the array 1 were, based on this pattern execution it looks like array 1 would be the encrypted ciphertext.

```
using (MemoryStream memoryStream = new MemoryStream(array))
{
    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, 0))
    {
        using (StreamReader streamReader = new StreamReader(cryptoStream))
        {
            if (streamReader.ReadToEnd() == this.SecretInput.Text)
            {
                this.SecretOutput.Text = "Congratz! You found the secret message";
            }
            else
            {
                this.SecretOutput.Text = "Sorry. Not correct password";
            }
        }
    }
}
```

Now we've the 3 array information, we can start crafting our cyberchef recipe & voila! We captured our flag!!



Now let's test and see the flag we found on the android app & it looks like that's the correct answer here



Flag: HTB{MyPasswordIsVerySecure}