

Challenge Walkthrough

Thursday, March 3, 2022 7:53 PM

The challenge information

The challenge information page for 'SeeTheSharpFlag'. The challenge is categorized as MEDIUM. It has NO CONNECTION REQUIRED. The challenge description is: 'I have made a password verification app. If I can remember the password, the app will tell me it is correct. See if you can guess my password.' The challenge rating is 135, and there is 1 comment. The user solves count is 439. The category is Mobile. The challenge was released 208 Days ago. Challenge creators are heartpoll & bertolis. A user named nobodyatall has completed the challenge. HTB-Bot achieved first blood.

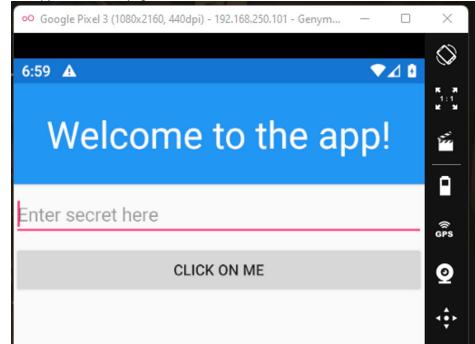
Try to sideload the app into android emulator

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

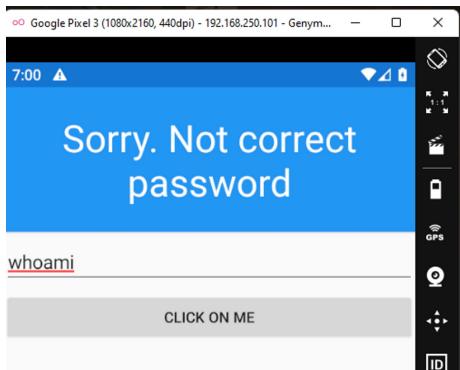
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Nameless> cd .\Desktop\
PS C:\Users\Nameless\Desktop> adb install .\com.companyname.seethesharpflag-x86.apk
Performing Streamed Install
Success
PS C:\Users\Nameless\Desktop>
```

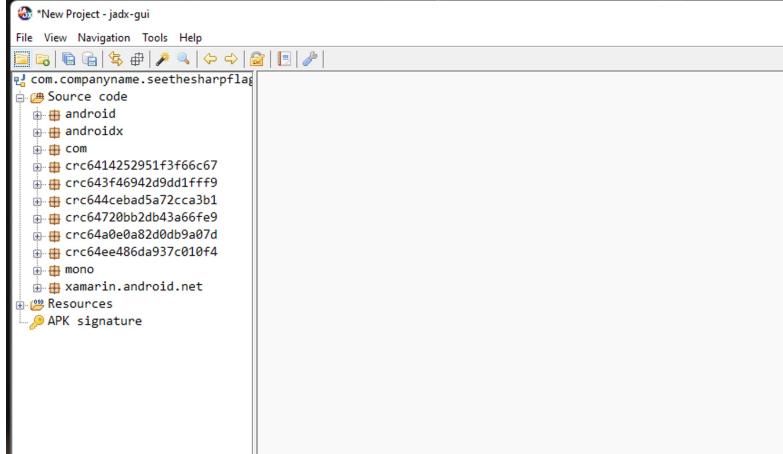
The application main page



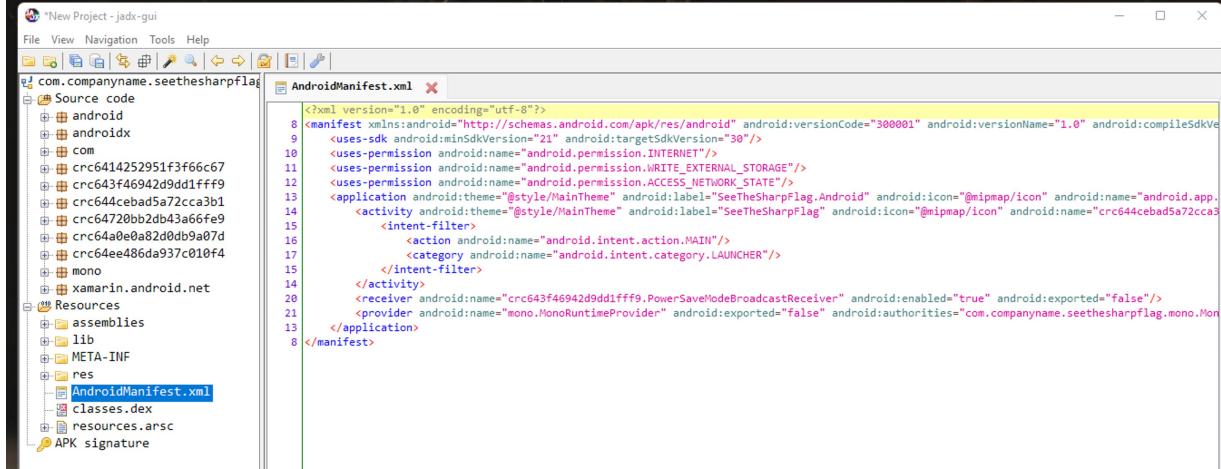
Type in random text & it shows wrong password



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 & class name 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" android:configChanges="orientation|screenLayout|uiMode">
    <activity android:theme="@style/MainTheme" android:label="SeeTheSharpFlag" android:icon="@mipmap/icon" android:name="crc644cebad5a72cca3b1.MainActivity" android:exported="false" android:category="LAUNCHER">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <receiver android:name="crc643f46942d9dd1ffff9.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  crc64cebad5a72cca3b1.MainActivity
package crc64cebad5a72cca3b1;

import android.os.Bundle;
import crc64f46942dd1ffff9.FormsAppCompatActivity;
import java.util.ArrayList;
import mono.android.IGCUserPeer;
import mono.android.Runtime;
import mono.android.TypeManager;

14 public class MainActivity extends AppCompatActivity implements IGCUserPeer {
    public static final String __md_methods = "n_onCreate:(Landroid/os/Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\nn_onRequestPermissionsResult:(I[Ljava/lang/String;[I)V:GetOnRequestPermissionsResult_IArrayHandler";
    private ArrayList refList;

    private native void n_onCreate(Bundle bundle);

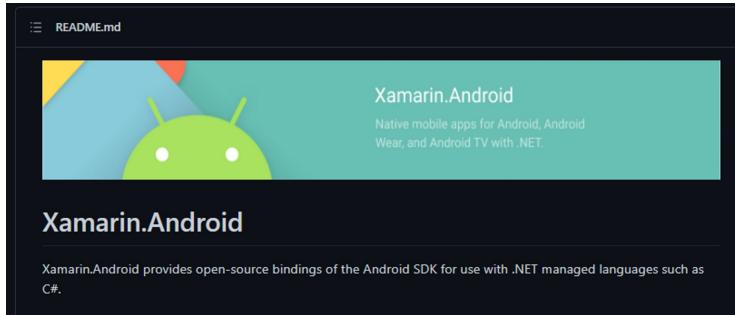
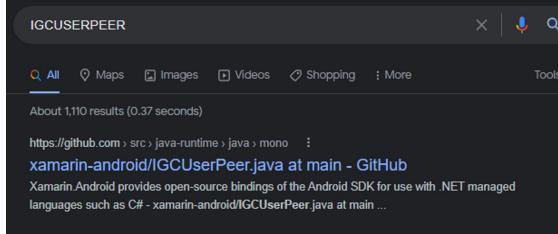
    private native void n_onRequestPermissionsResult(int i, String[] strArr, int[] iArr);

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

21     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

```

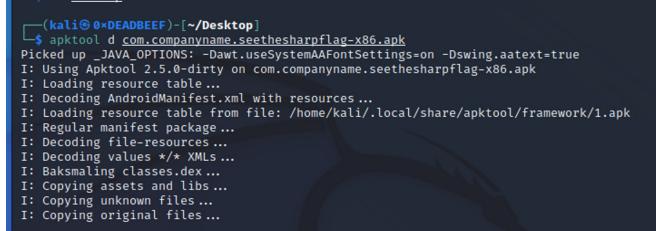
static {
    Runtime.register("SeeTheSharpFlag.Droid.MainActivity", SeeTheSharpFlag.Android", MainActivity.class, __md_methods);
}

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

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

```

Now use apktool to decompress the apk



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



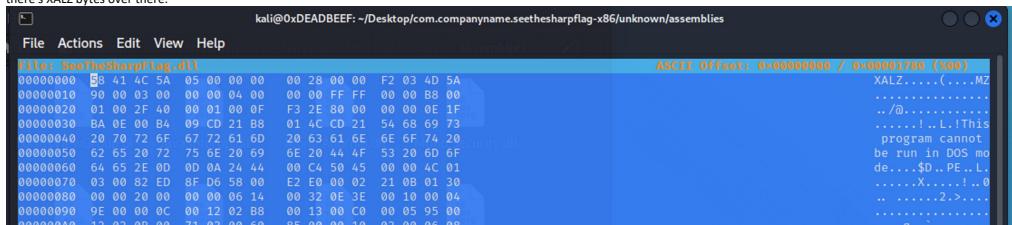
In the directory, noticed that there's bunch of dll files and some of them are xamarin dll.

```
(kali㉿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.dll  Xamarin.AndroidX.SavedState.dll
Mono.Security.dll           Xamarin.AndroidX.AppCompatResources.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㉿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

["xamarin" "XALZ"](#)

About 68 results (0.38 seconds)

<https://www.x41-dsec.de/xamarin-dll-decompression> :

Decompressing Xamarin DLLs | X41 D-SEC

22 Sept 2020 — After searching for the magic file header XALZ, we found the public Xamarin pull request from May 2020 that described the new file format.

https://github.com/blob/master/Mobile/Xamarin/tools/Xamarin_XALZ_decompress.py at master · x41sec/tools

<https://github.com/xamarin/xamarin-android/pull/4686>. Installation notes: This program requires the python lz4 library. Install via. "lz4" (pip).

<https://github.com/NickstaDB/xamarin-decompress> :

NickstaDB/xamarin-decompress - GitHub

7 Oct 2020 — The script checks for the XALZ header which indicates a compressed assembly and decompresses it using LZ4 block decompression so that the ...

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.

github.com/x41sec/tools/blob/master/Mobile/Xamarin/Xamarin_XALZ_decompress.py

Search or jump to... Pull requests Issues Marketplace Explore

/ tools Public

Issues 1 Pull requests Actions Projects Wiki Security Insights

master tools / Mobile / Xamarin / Xamarin_XALZ_decompress.py / <> Jump to ... Go to file ...

Christian Reitter (X41) rename path Latest commit c3abfe on Sep 21, 2020 History

At 0 contributors

Executable File | 74 lines (55 sloc) | 2.61 KB

Raw Blame Copy Edit Delete

Now try to decompress the DLL file with Xamarin_XALZ_decompress.py

```

File Actions Edit View Help x86

(kali㉿DEADBEEF) [~/Desktop]
$ python3 XamarXALZ_decompress.py com.companyname.seethesharpflag-x86/unknown/assemblies/SeeTheSharpFlag.Android.dll SeeTheSharpFlag.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㉿DEADBEEF) [~/Desktop]
$ python3 XamarXALZ_decompress.py com.companyname.seethesharpflag-x86/unknown/assemblies/SeeTheSharpFlag.dll SeeTheSharpFlag_Unpack.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㉿DEADBEEF) [~/Desktop]
$ 

```

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

```

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

(kali㉿DEADBEEF) [~/Desktop]
$ 

```

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

```

dnSpy v6.1.7 (64-bit)
File Edit View Debug Window Help C# Start Search

Assembly Explorer SeeTheSharpFlag.Android (1.0.0.0)
1 // C:\Users\Malwally\Desktop\SeeTheSharpFlag.Android_Unpack.dll
2 // SeeTheSharpFlag.Android, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
3
4 // Timestamp: <Unknown> (AB8805BC)
5
6 using System;
7 using System.Diagnostics;
8 using System.Reflection;
9 using System.Runtime.CompilerServices;
10 using System.Runtime.InteropServices;
11 using System.Runtime.Versioning;
12 using Android.App;
13 using Android.Runtime;
14
15 [assembly: AssemblyVersion("1.0.0.0")]
16 [assembly: CompilationRelaxations(8)]
17 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
18 [assembly: Debuggable(true)]
19 [assembly: ResourceDesigner("SeeTheSharpFlag.Droid.Resource", IsApplication = true)]
20 [assembly: AssemblyTitle("SeeTheSharpFlag.Android")]
21 [assembly: AssemblyDescription("")]
22 [assembly: AssemblyConfiguration("")]
23 [assembly: AssemblyCompany("")]
24 [assembly: AssemblyProduct("SeeTheSharpFlag.Android")]
25 [assembly: AssemblyCopyright("(Copyright © 2014)")]
26 [assembly: AssemblyTrademark("")]
27 [assembly: ComVisible(false)]
28 [assembly: AssemblyFileVersion("1.0.0.0")]
29 [assembly: UsesPermission("android.permission.INTERNET")]
30 [assembly: UsesPermission("android.permission.WRITE_EXTERNAL_STORAGE")]
31 [assembly: TargetFramework("MonoAndroid,Version=v11.0", FrameworkDisplayName = "Xamarin.Android v11.0 Support")]
32

```

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.

```

File Edit View Debug Window Help C# Start Search

Assembly Explorer MainPage
1 using System;
2 using System.CodeDom.Compiler;
3 using System.IO;
4 using System.Reflection;
5 using System.Security.Cryptography;
6 using Xamarin.Forms;
7 using Xamarin.Forms.Internals;
8 using Xamarin.Forms.Xaml;
9 using Xamarin.Forms.Xaml.Diagnostics;
10 using Xamarin.Forms.Xaml.Internals;
11
12 namespace SeeTheSharpFlag
13 {
14     // Token: 0x02000003 RID: 3
15     [XmlFilePath("MainPage.xaml")]
16     public class MainPage : ContentPage
17     {
18         // Token: 0x06000007 RID: 7 RVA: 0x000002135 File Offset: 0x00000335
19         public MainPage()
20         {
21             this.InitializeComponent();
22         }
23
24         // Token: 0x06000008 RID: 8 RVA: 0x000002144 File Offset: 0x00000344
25         private void Button_Clicked(object sender, EventArgs e)
26         {
27             byte[] array = Convert.FromBase64String("sjAbajc4sWUn6CHjBSf039p2fNg2trNQ/MmTB5mn0=");
28             byte[] array2 = Convert.FromBase64String("6FH+HgZepQXodJv-i7l14o==");
29             byte[] array3 = Convert.FromBase64String("Dz6YdahJ1Zav2GvNEEQ31A==");
30             using (AesManaged aesManaged = new AesManaged())
31             {
32                 using (ICryptoTransform cryptoTransform = aesManaged.CreateDecryptor(array2, array3))
33                 {
34                     using (MemoryStream memoryStream = new MemoryStream(cryptoTransform))
35                     {
36                         using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, 0))
37                         {
38                             using (StreamReader streamReader = new StreamReader(cryptoStream))
39                             {
40                                 if (streamReader.ReadToEnd() == this.SecretInput.Text)
41                                 {
42                                     this.SecretOutput.Text = "Congratz! You found the secret message";
43                                 }
44                                 else
45                                 {
46                                     this.SecretOutput.Text = "Sorry. Not correct password";
47                                 }
48                             }
49                         }
50                     }
51                 }
52             }
53         }
54     }
55 }

```

The encryption that this password used are AES

```

using (AesManaged aesManaged = new AesManaged())
{
    ...
}

```

```
https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aesmanaged?view=dotnet-10.0
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_Click(object sender, EventArgs e)
{
    byte[] array = Convert.FromBase64String("sjAbajc4sWUn6CHJBSfQ39p2fNg2trIVQ/MmTB5mno=");
    byte[] array2 = Convert.FromBase64String("6F+wgzEp5QXodJV+iTl14Q==");
    byte[] array3 = Convert.FromBase64String("DZ6YdaWJ1Zav26VmEEQ31A==");
    using (AesManaged aesManaged = new AesManaged())
    {
        ...
    }
}
```

The first part it used aesManaged.CreateDecryptor() with array2 & array3 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.

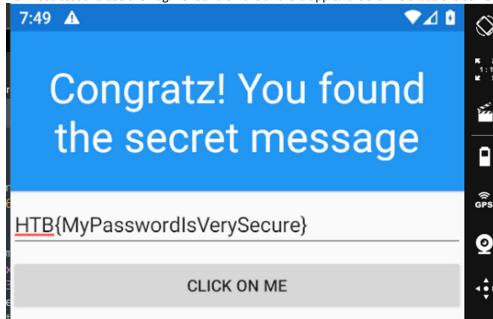
```
using (MemoryStream memoryStream = new MemoryStream(array))
{
    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, 0))
    {
        using (StreamReader streamReader = new StreamReader(cryptoStream))
        {
            if (streamReader.ReadToEnd() == this.SecretInput.Text)
            {
                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!

Last build: 6 months ago

Recipe	Input	
From Base64	sjAbajc4sWUn6CHJBSfQ39p2fNg2trIVQ/MmTB5mno=	
Alphabet A-Za-z0-9+=		
<input checked="" type="checkbox"/> Remove non-alphabet chars		
AES Decrypt		
Key 6F+wgzEp5QXodJV+iTl14Q==	BASE64	
IV DZ6YdaWJ1Zav26VmEEQ31A==	BASE64	
Mode CBC	Input Raw	Output Raw
Output		
HTB{MyPasswordIsVerySecure}		

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}