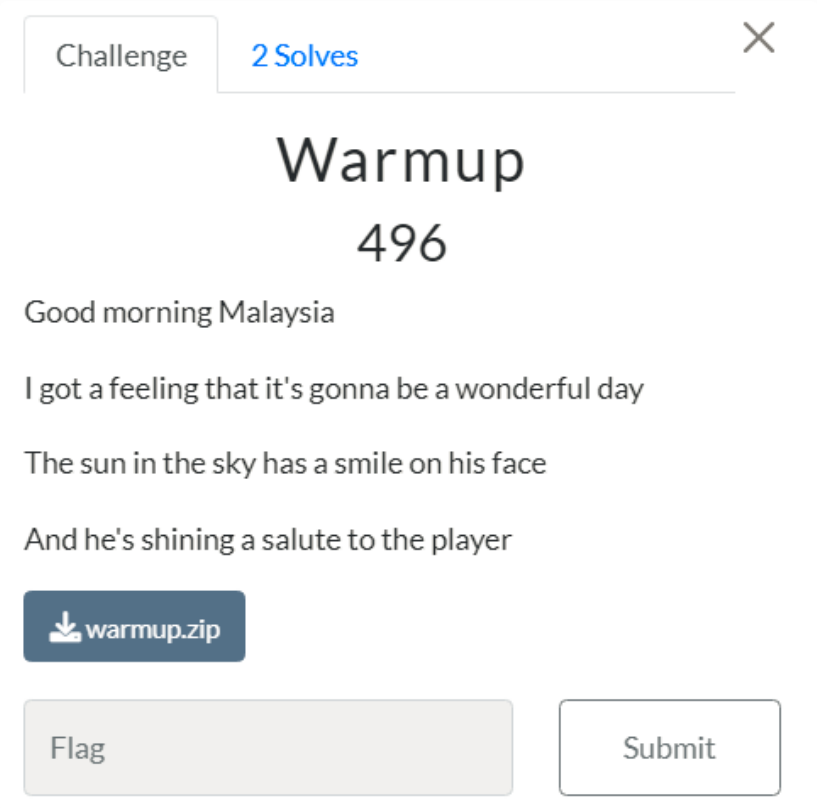
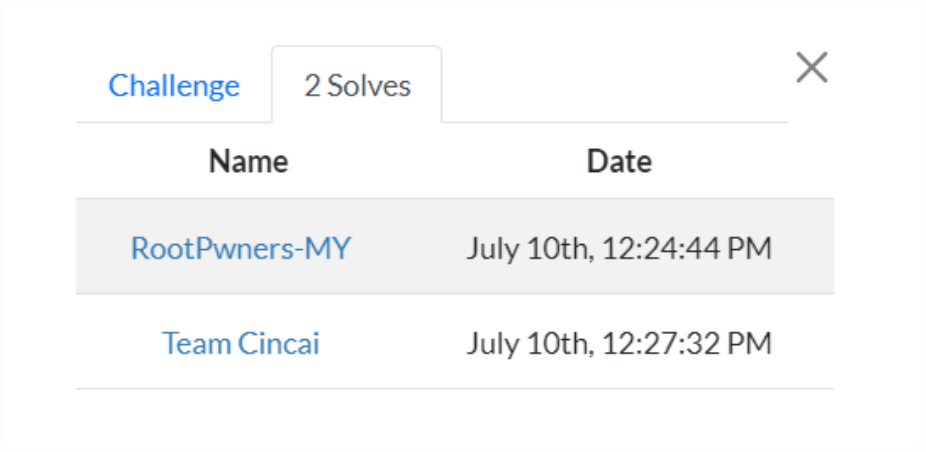
CYDES 2023: Manual Unpacking MPRESS(2.19) - The ESP Trick

Before we get started, I would like to take a moment to thankyou the organizers of the Cyber Warzone CTF challenges. The National Cyber Security Agency Malaysia (NACSA), Velum Labs, etc and the technical team WargamesMY have all worked hard to make this event success.

Warmup



I spend many hours figuring it out, lol. The challenge can be solve with easy way, i choose the hard road. I love myself.

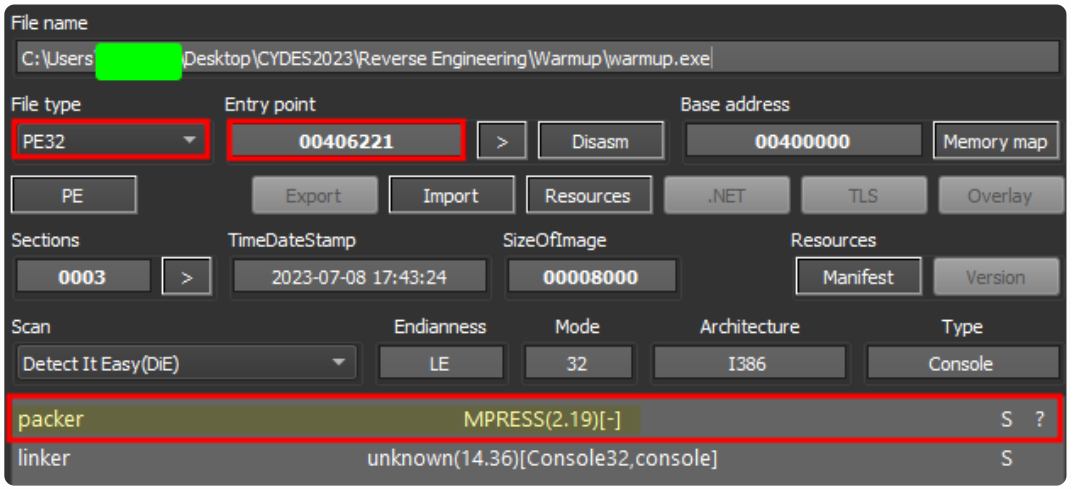


Dynamic Analysis

```
Hi, here for some warmup?

Please enter the flag
>cookies
Uh-oh, bad warmup can cause injury.
```

This is a clear indicator that the program is packed.



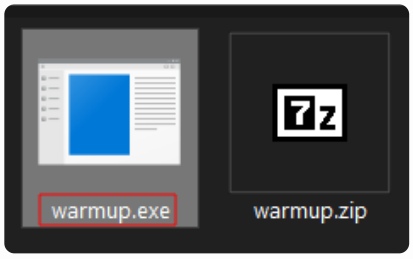
By looking at the sample, we know its PEx32 file type and the entry point is 00406221 and the file is packed. The plan here is that, we need to unpacked it first, but how? should we just find any public automated tools? or can we unpacked it manually?

So, the answer is that we will do it manually. Because, why not? :)

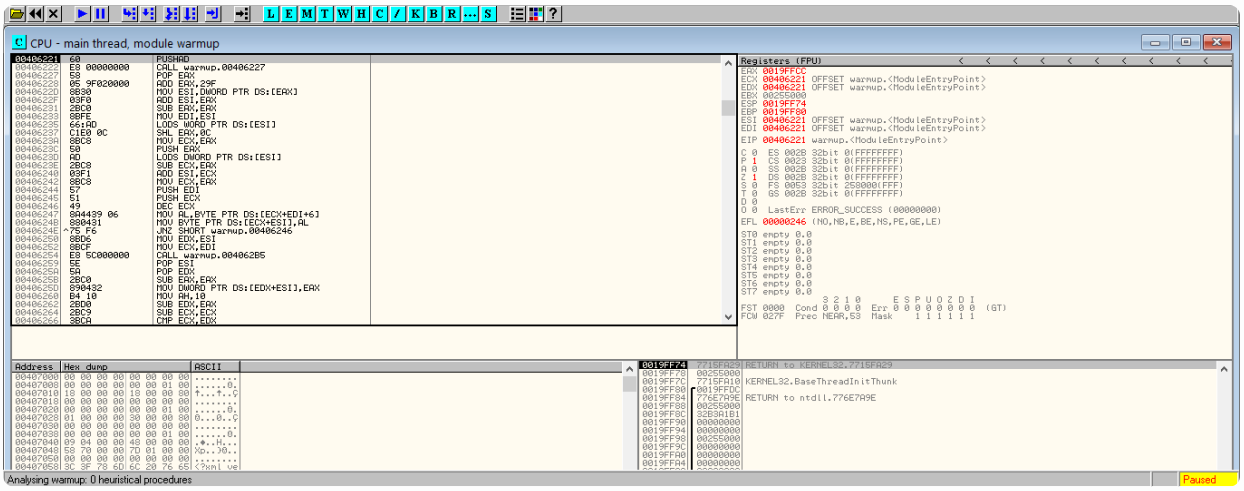
Unpacking MPRESS(2.19) - The ESP Trick

In order to successfully reverse engineer packed we need to debug it until we get to the decompressed memory section. Then we can dump that out and analyze that dumped executable.

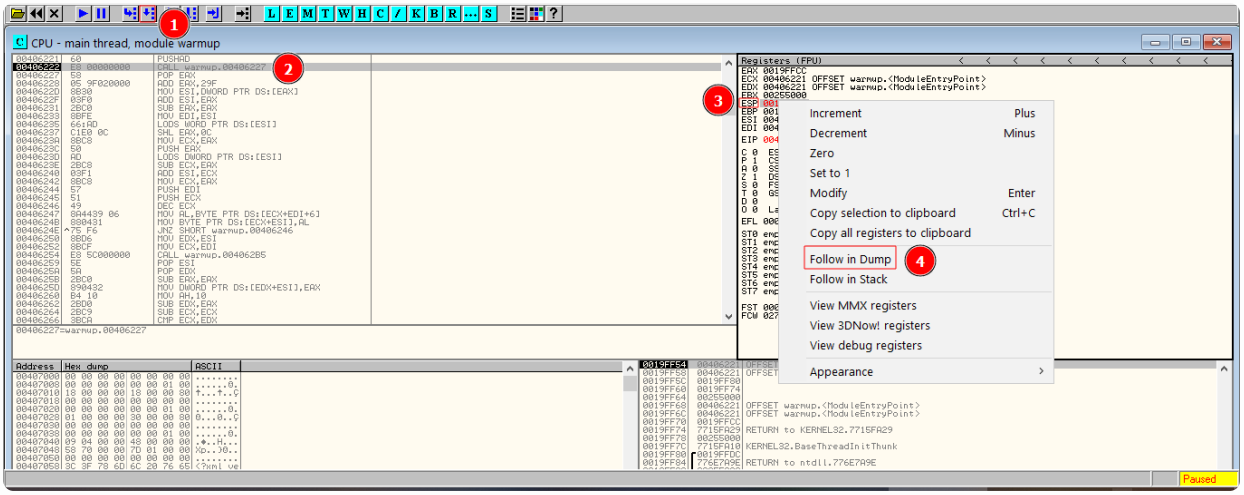
One trick in doing that is the “ `ESP` trick”. So named for the `ESP` register, we can use this trick to set a hardware breakpoint on the `ESP` register, and when we get to the breakpoint we should be at the Original Entry Point (`OEP`) of the program. We can then dump the rest of the executable and we should have our unpacked executable.



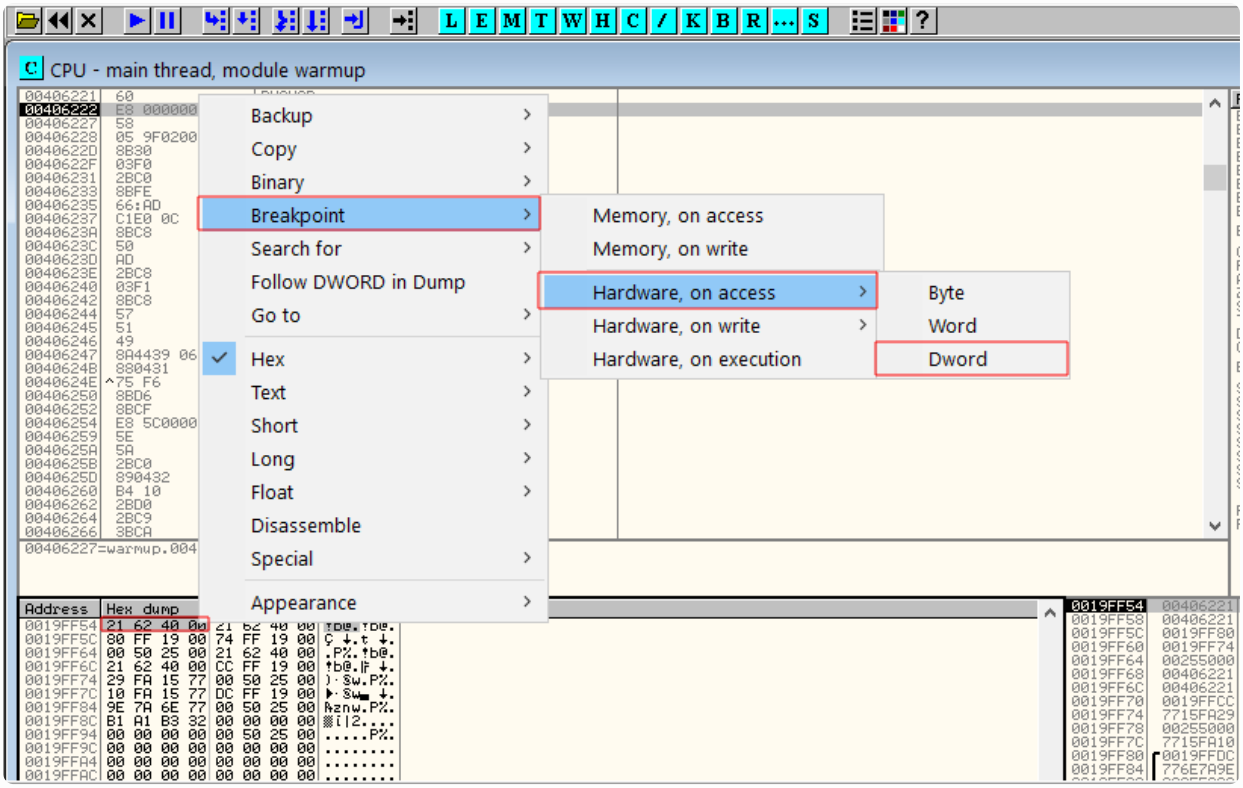
Load the sample to any debugger that you can use, make sure it support x32. Run as admin to get full access on it.



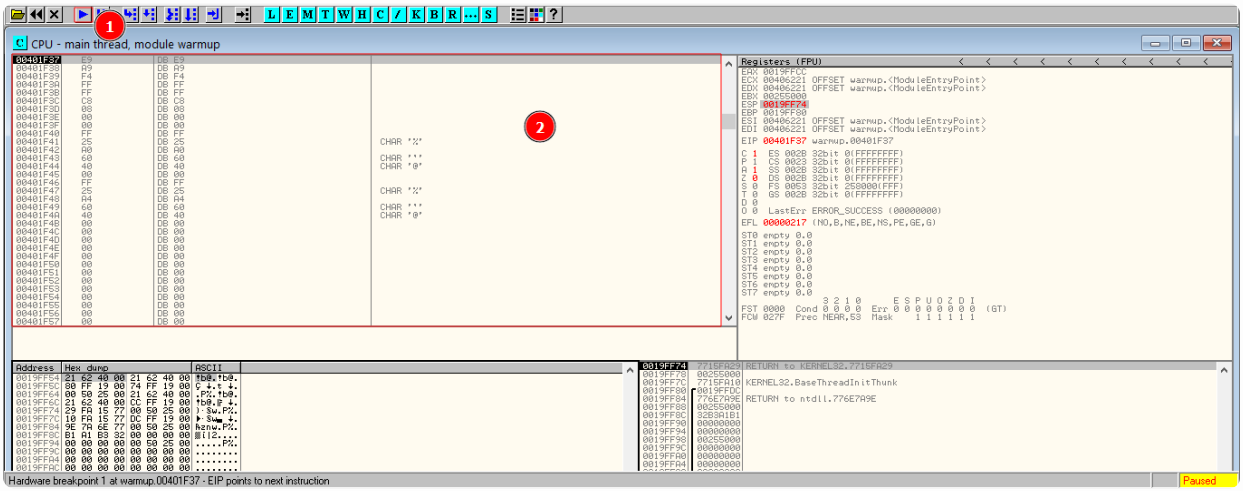
Step over to the call



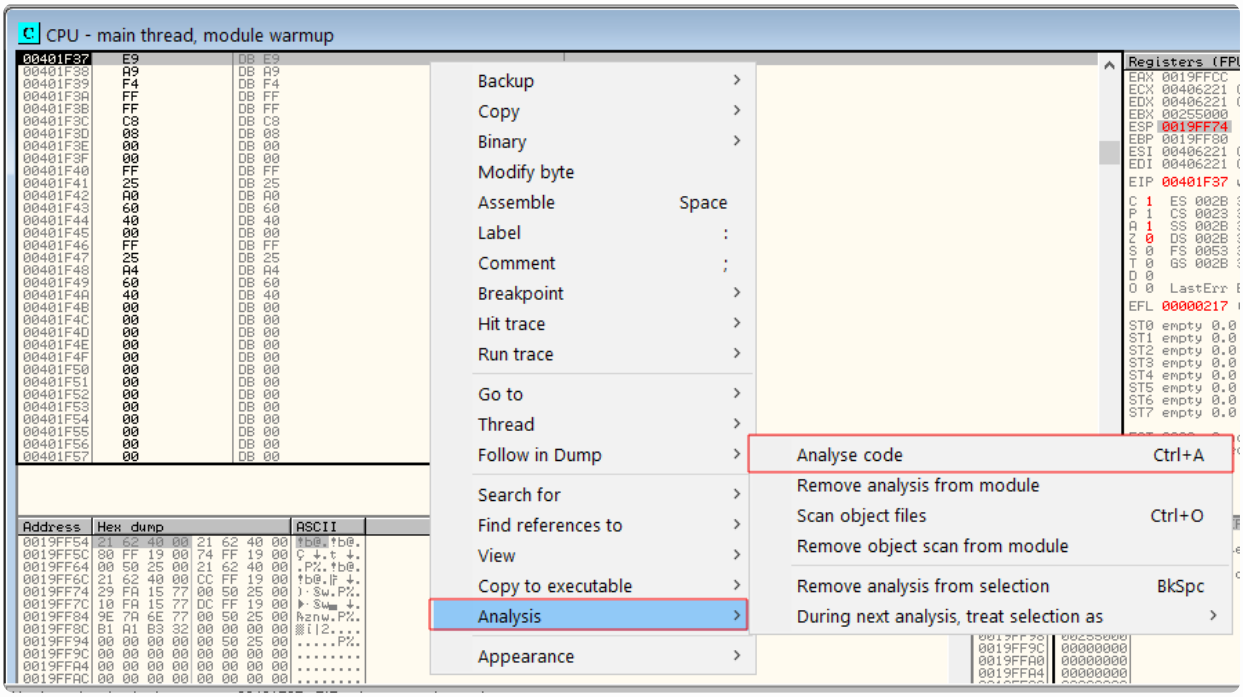
Then we select the first four bytes in the dump at the bottom of debugger and set a hardware access breakpoint on the DWord. This will have us break right before we unpack the executable.



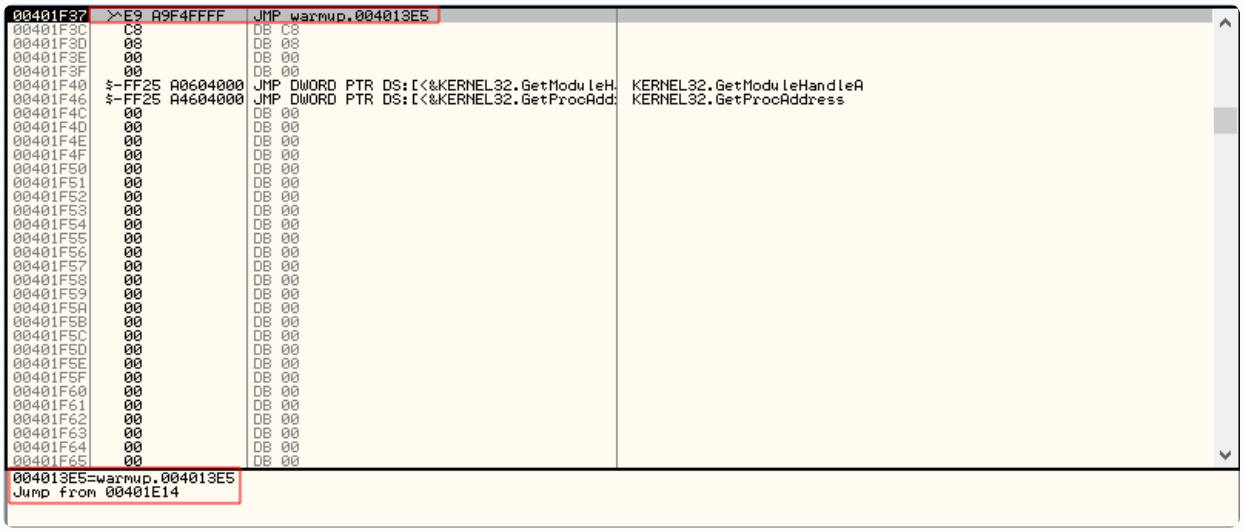
and click run, it will stop at the breakpoint



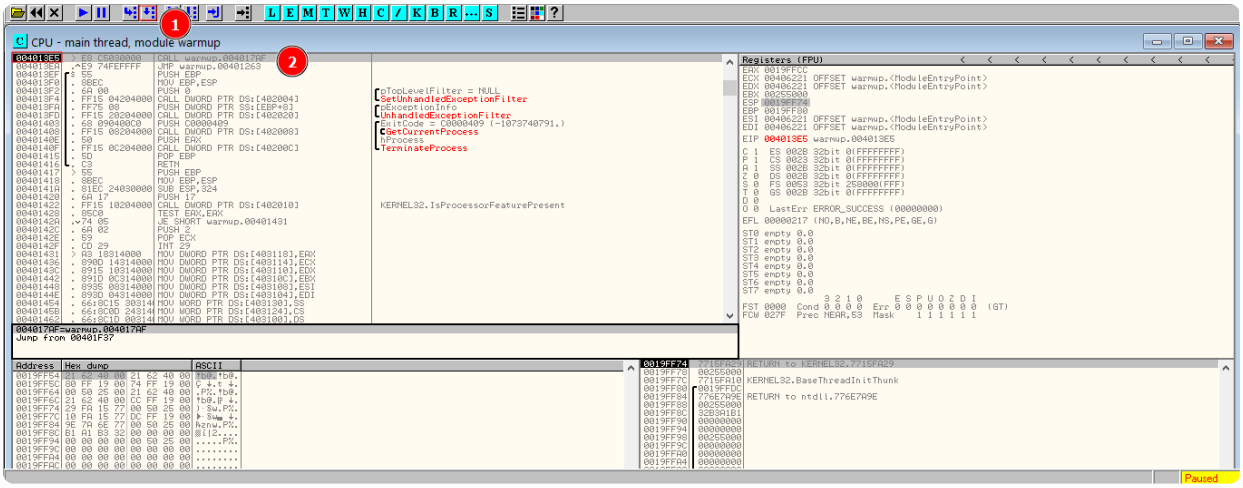
next, you can analyst the code



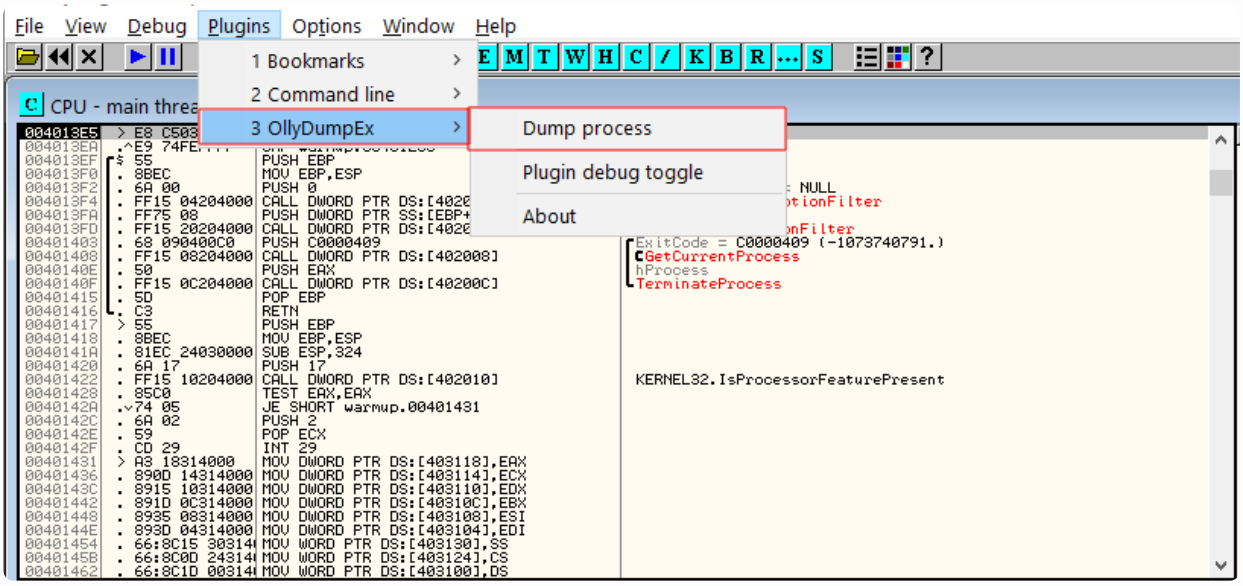
It's unconditional JMP, it will jump to address warmup.004013E5



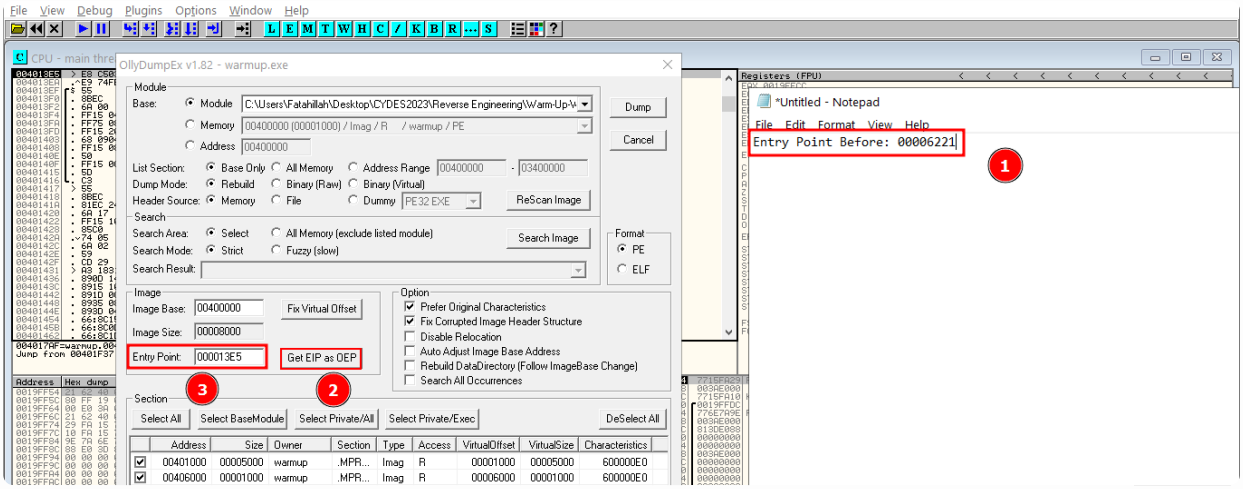
Check the address, make sure its on the right place



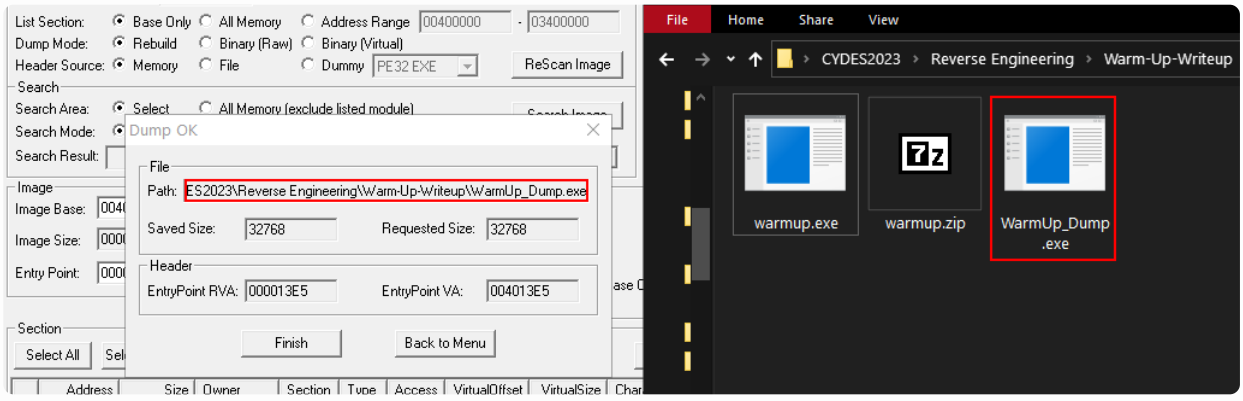
We can use OllyDumpEx to Dump the process



Copy the Entry point before and put it somewhere note, next click to get the current EIP and dump it

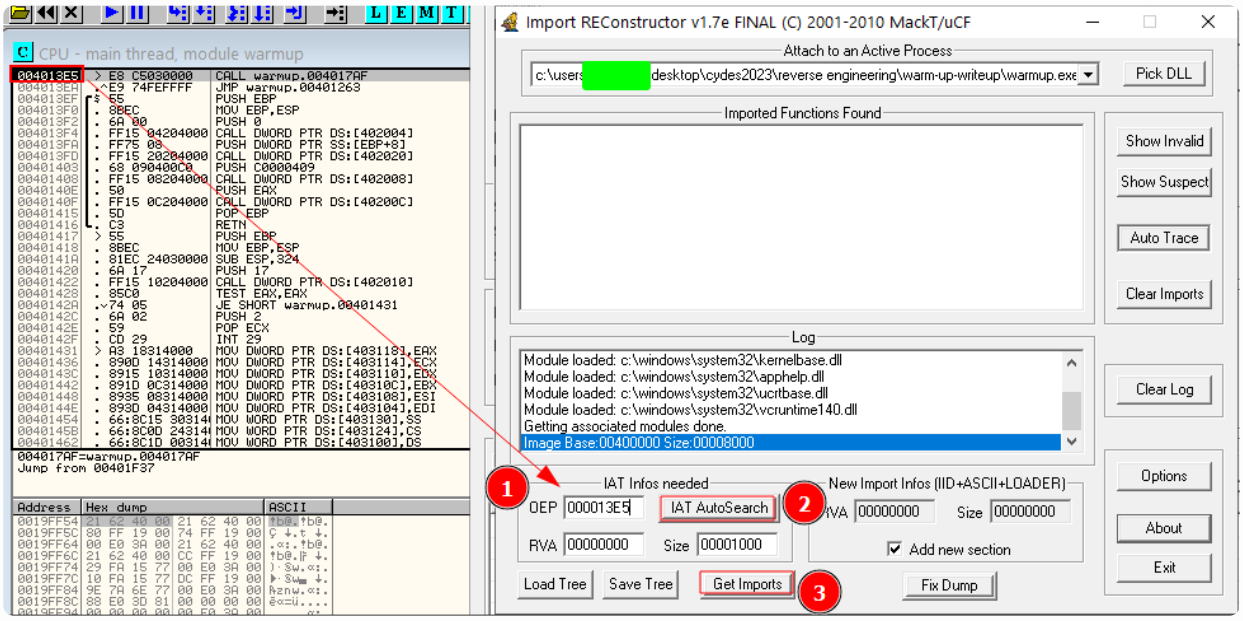


Save the file(WarmUp_Dump.exe).

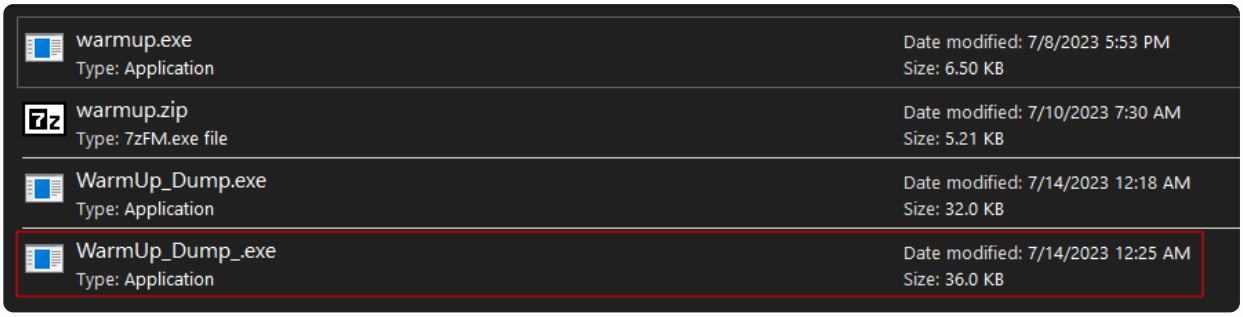


Get the current EIP and click IAT Auto Search to automatically find the Import Address Table of the executable. After that click “Get Imports” to get a list of the imports that the

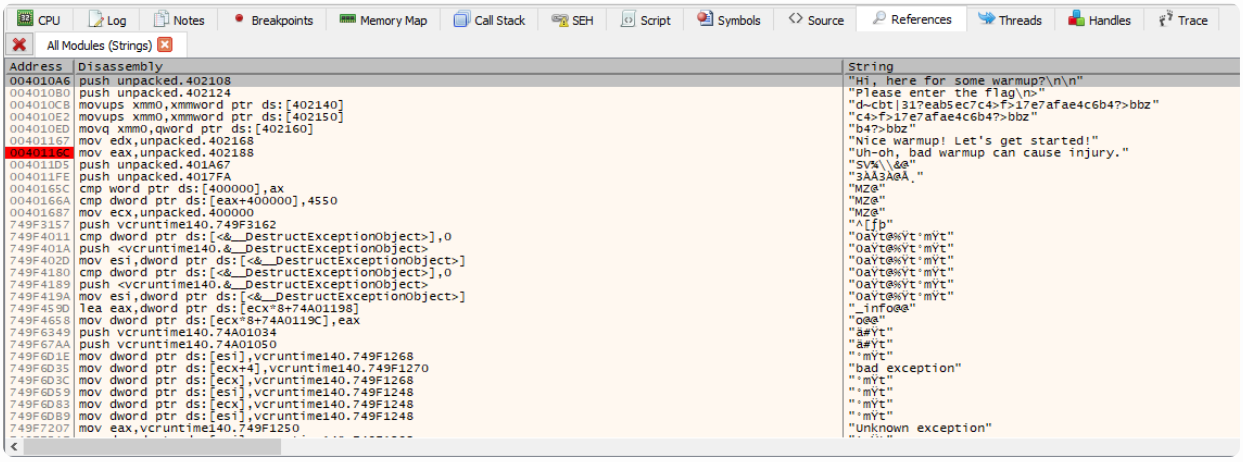
executable has. Choose the dump file and fix it.



This is the unpacked version, look at the size. Rename it to make things easier.

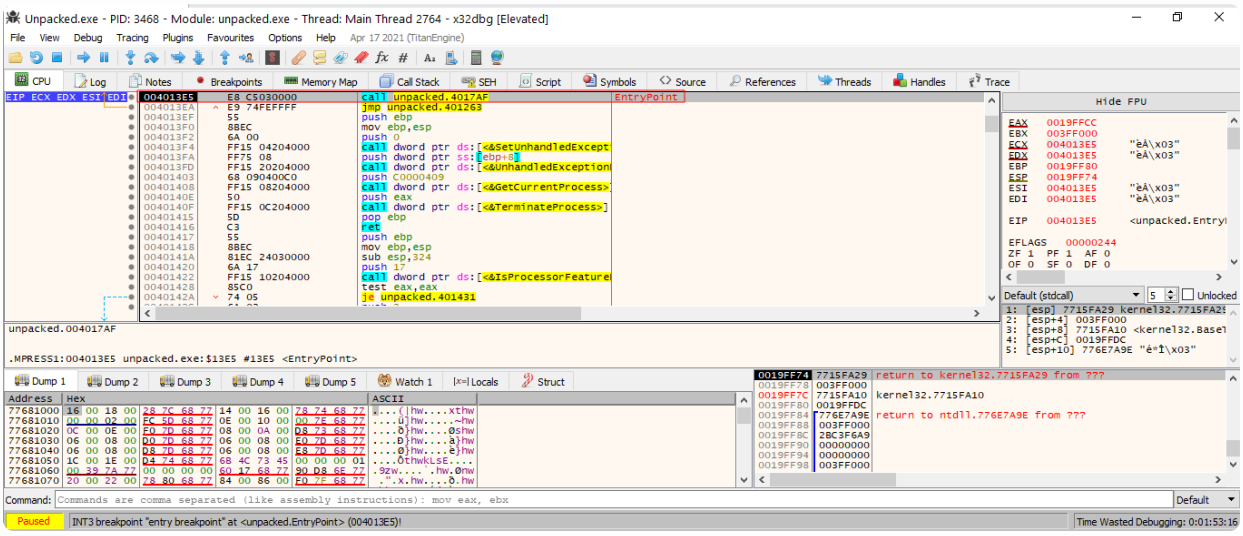


Now everything looks clear on the strings, we succesfully unpacked the sample.

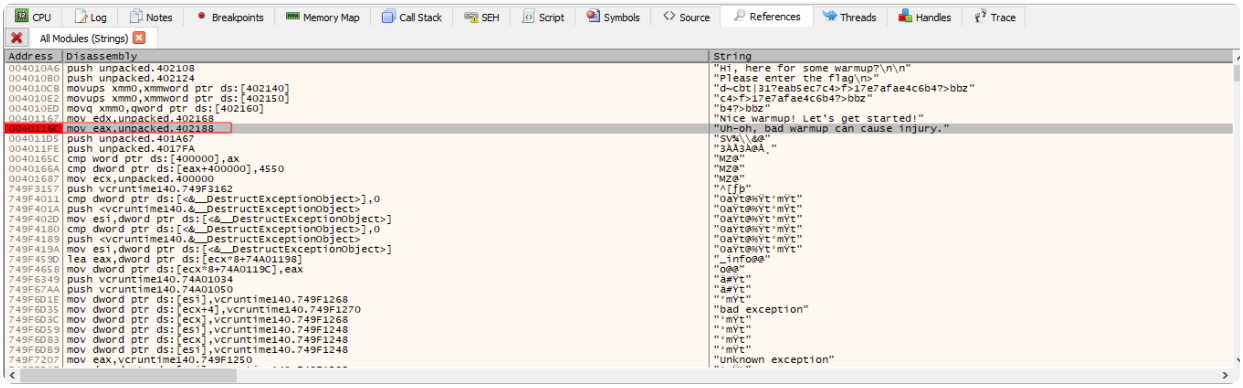


Next, we just need to crack the software and get the flag.

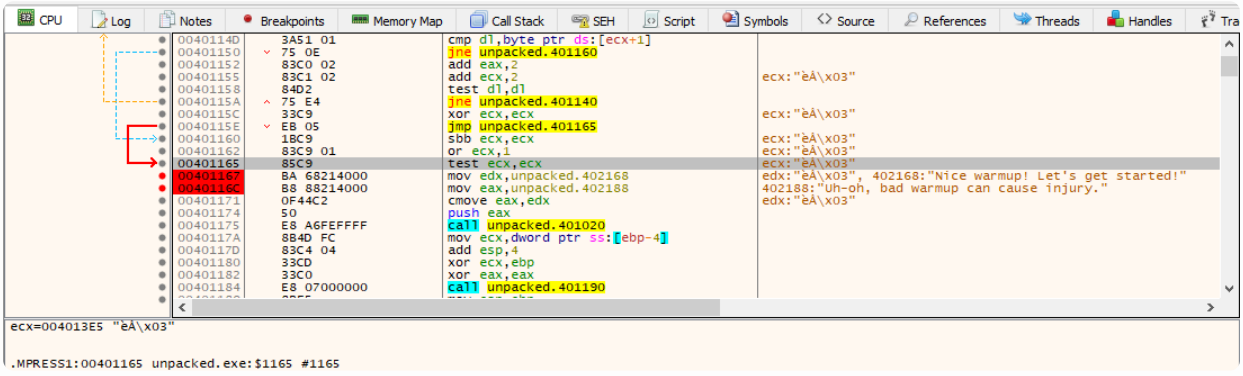
Run and go to breakpoint software



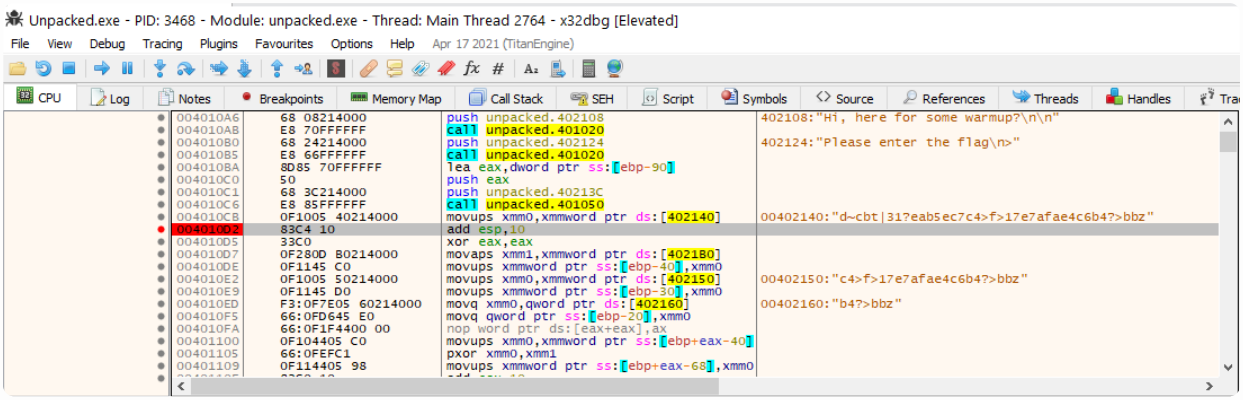
Go to the false instruction address



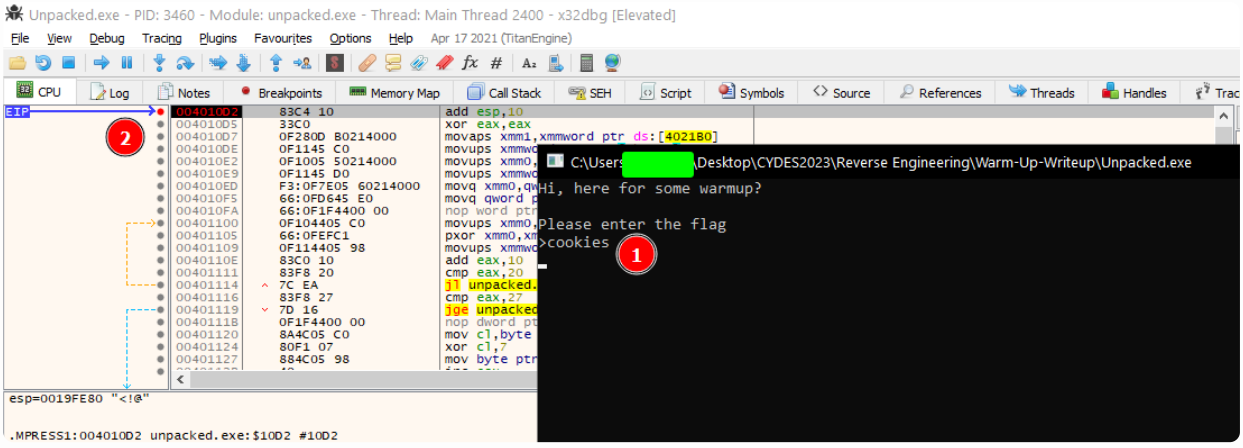
The comparison will be happen on the top



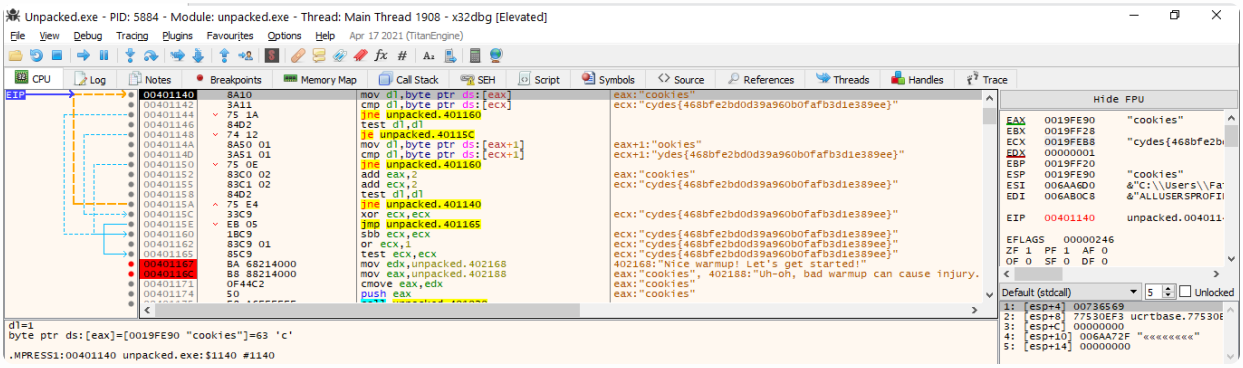
Do analyze the instruction carefully



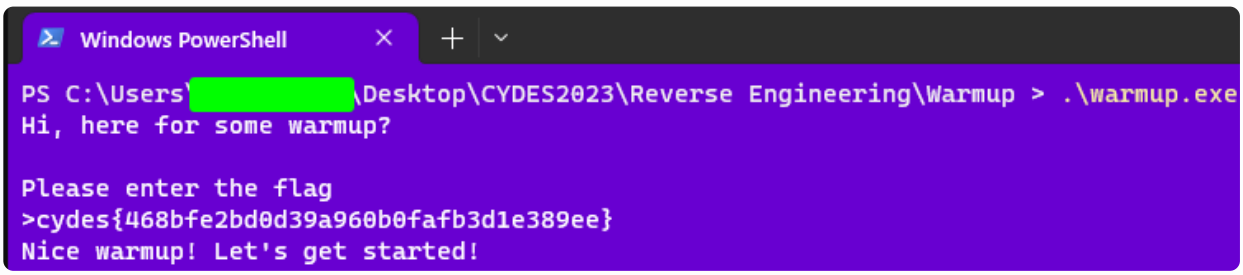
The false will break at this address, keep step over.



You will find instructor that compare between ECX and EAX. This is where comparison is made.



That's the Flag.



Ok challenge. but I'm surprised it only got 2 solves! I think rev/pwn tend to scare people away sadly.

`cydes{468bfe2bd0d39a960b0fafb3d1e389ee}`

Power of Rewind

This one got the code, i just brute force(AI) the code until i get the flag, Because i have no time to waste.

```
$base64String =
"FgJqAMKJ5ePgsWMLneXHLrXKhmjNwCYUDCpD3u8sbiT8sEJ9M1GmdzrYkXP64PYv"

$encryptedBytes = [System.Convert]::FromBase64String($base64String)

$key = 145,96,34,150,165,222,211,99,165,119,17,98,225,14,249,255
$iv = 251,122,202,111,165,48,247,134,32,88,101,199,33,154,190,56

$aes = New-Object System.Security.Cryptography.RijndaelManaged
$aes.Mode = [System.Security.Cryptography.CipherMode]::CBC
$aes.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
$aes.Key = $key
$aes.IV = $iv

$decryptor = $aes.CreateDecryptor()

$decryptedBytes = $decryptor.TransformFinalBlock($encryptedBytes, 0,
$encryptedBytes.Length)
$decryptedString = [System.Text.Encoding]::UTF8.GetString($decryptedBytes)

$decryptedString
```

`cydes{ce65c25c5bd0fa669bd3bdef7aa9bdac}`