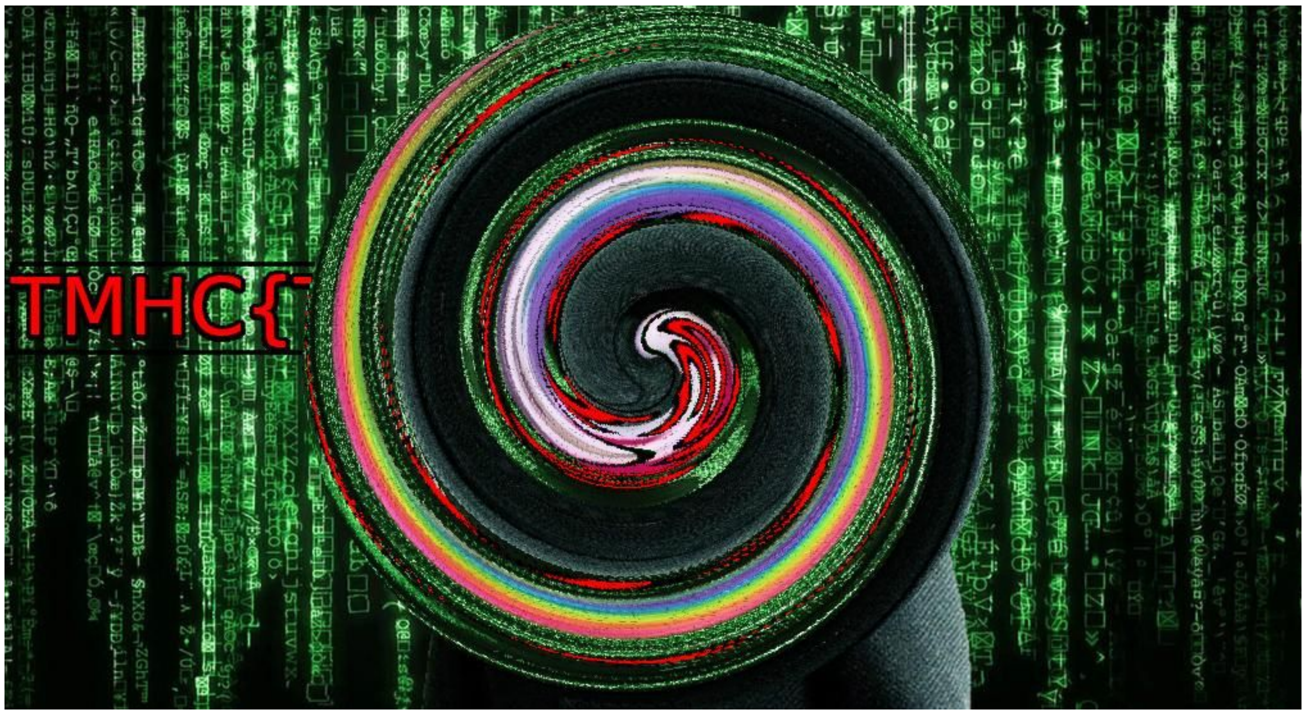
WANTED! Writeup



1) Implementing the required steps one by one

Effect starts by determining the largest circle originated in the image's center.

```
#get largest contained circle radius
radius = 0
if(imagex>imagey): radius=int(round(imagey/2))
else: radius = round(imagex/2)
```

Then, each pixel inside this circle is rotated linearly depending on how far away
from the sides original pixel is.
While pixels just on the edge stay the same, pixels r/4 away are rotated 90 degrees, halfway are rotated 180 degrees, until pixels in the center are rotated 360 degrees for a single twirl.

Swirl amount as degree per point:

```
def getSwirlDegree(point,radius): #get degrees to rotate per point
        px,py = point
        dist = round(math.sqrt(px*px + py*py))
        #print(dist)
        if(dist>radius):
                return -1
        else:
                return 360.0 * ((radius - dist)/radius)
```

However, our analysis shows filter was run twice, so you'll have to double the
values (r/4->180 degrees, r/2 -> 360 degrees, center 720)

We can just add a variable called numswirls so its easy to tweak with, in our case numswirls will be 2.

```
def getSwirlDegree(point,radius): #get degrees to rotate per point
        px,py = point
        dist = round(math.sqrt(px*px + py*py))
        #print(dist)
        if(dist>radius):
                return -1
        else:
                return 360.0 * getSwirlDegree.numswirls * ((radius - dist)/radius)
```

2) **With above conditions met, we can continue on implementing a swirl.**

Loading the image as an np array:

```
def loadImage(infilename):
    img = Image.open(infilename)
    data = np.array(img)
    return data
```

Following function will rotate a coordinate by given degrees, around origin. *(direction 1 will will swirl again, while direction -1 will unswirl)*

```
def rotateCoo(point,degree): #rotate a coordinate by 
degrees
        if(degree==-1): #out of scope
                return (-1,-1)
        px,py = point
        angle = rotateCoo.direction * math.radians(degree) 
#degree to radian
        _x = math.cos(angle) * px - math.sin(angle) * py
        _y = math.sin(angle) * px + math.cos(angle) * py

        #print(str(point) + " -> " + str((_x,_y)) + ": " + 
str(degree))
        return (int(round(_x)),int(round(_y)))
```

However, we don't want to rotate around (0,0). We want to rotate around image center. To keep the math simple, this can be done by offsetting the coordinate by image center before and after the rotation.

Setting up an offset/unoffset function:

```
im = loadImage(inf)
imagex = len(im)
imagey = len(im[0])
_offset= (int(round(imagex/2)), int(round(imagey/2)))
offsetPoint.offset=_offset
def offsetPoint(point, unoffset=False): #offset point to center
	if(unoffset): return (point[0] + offsetPoint.offset[0], point[1] +
offsetPoint.offset[1])
	else: return (point[0] - offsetPoint.offset[0], point[1] - offsetPoint.offset[1])
```

Mapping the new coordinates:

```
for i in range(0,imagex):
	for j in range(0,imagey):
		realPoint= offsetPoint((i,j))
		newCoords = rotateCoo(realPoint,getSwirlDegree(realPoint,radius))
		if(newCoords[0] == -1 and newCoords[1] == -1):
			result[i,j,0] = im[i][j][0]
			result[i,j,1] = im[i][j][1]
			result[i,j,2] = im[i][j][2]
			continue
		newCoords = offsetPoint(newCoords,True)
```

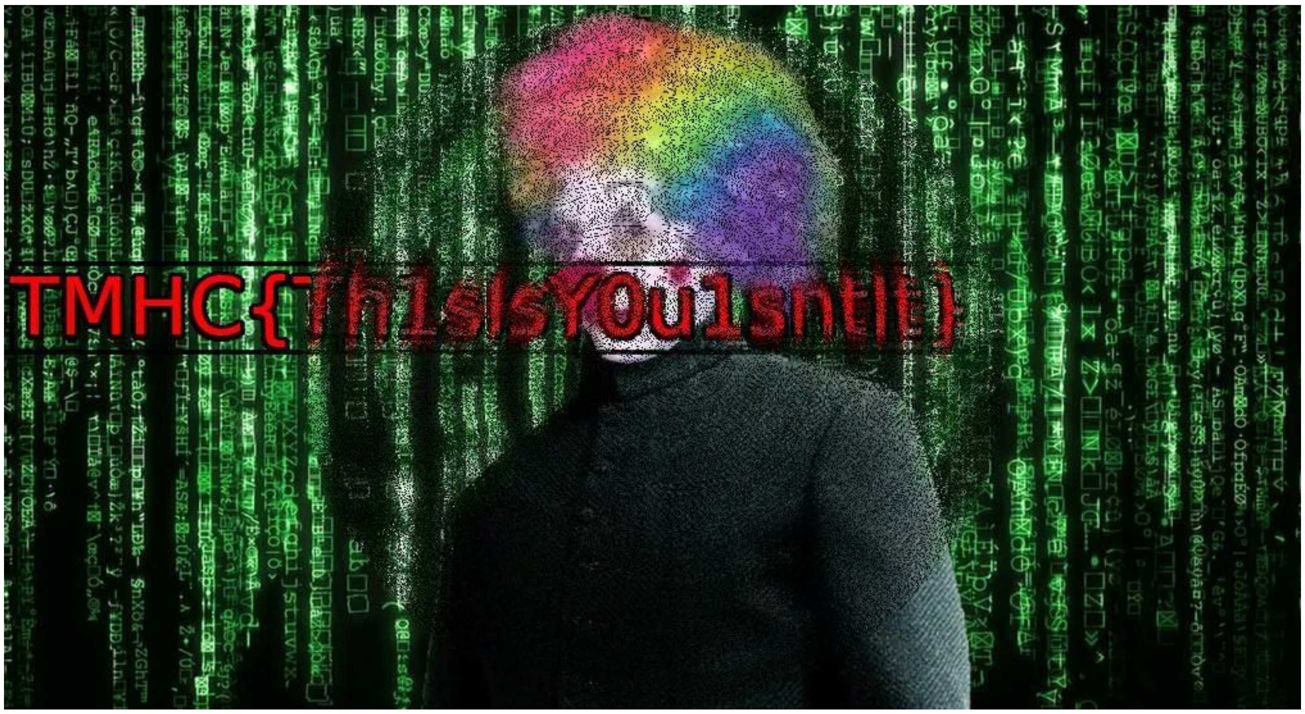
Optional: There will be empty pixels during swirling/unswirling due to floating errors and unmapped pixels in the result. These can be blurred with a simple function to increase readability.

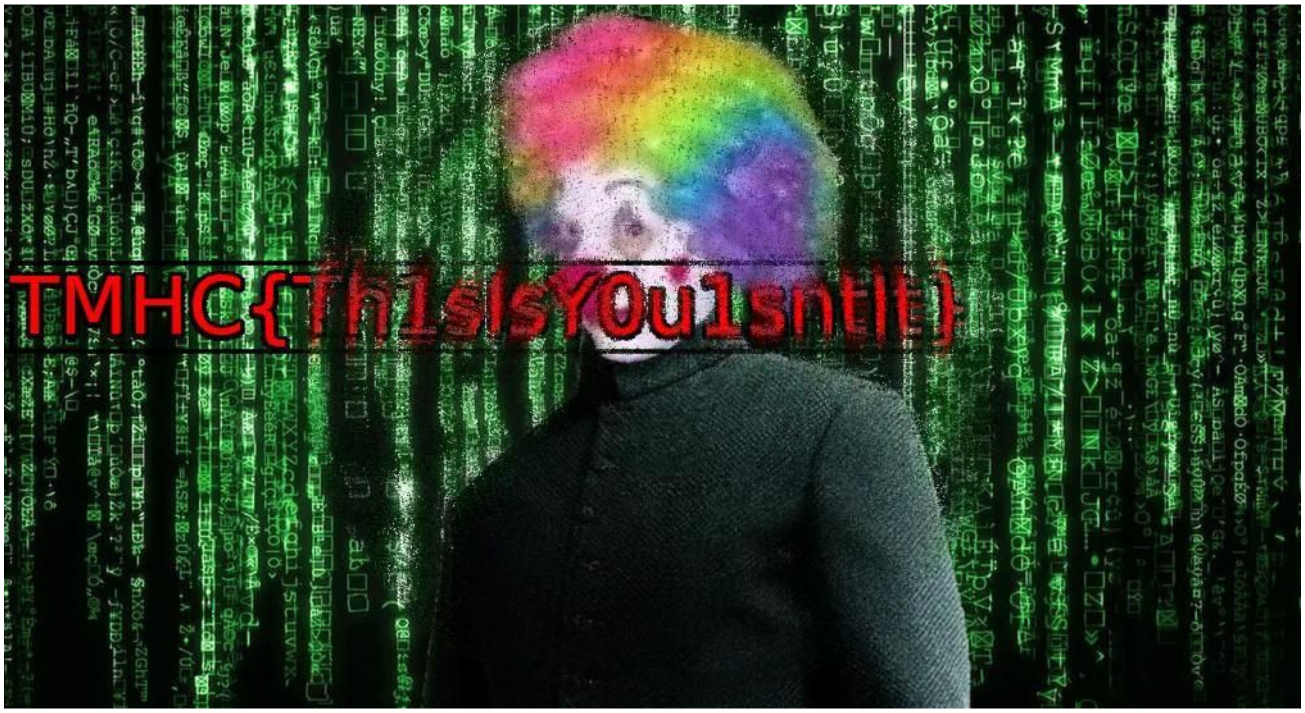
```
#Blur missing points in result image, lost to floating error.
for i in range(0,a-1):
	for j in range(0,b-1):
		for k in range(0,c):
			if(result[i][j][k]==0):
				#print("Found floating precision error. Blurring")
result[i,j,k]=(int(result[i+1,j,k])+int(result[i-1,j,k])+int(result[i,j+1,k])+int(result[i,j-1,
k]))/4
```

Final code:

```
import math
from PIL import Image
import numpy as np

inf =input("source:")
outf=input("target:")

def rotateCoo(point,degree): #rotate a coordinate by degrees
        if(degree==-1): #out of scope
                return (-1,-1)
        px,py = point
        angle = rotateCoo.direction * math.radians(degree) #degree to radian
        _x = math.cos(angle) * px - math.sin(angle) * py
        _y = math.sin(angle) * px + math.cos(angle) * py

        #print(str(point) + " -> " + str((_x,_y)) + ": " + str(degree))
        return (int(round(_x)),int(round(_y)))

rotateCoo.direction = int(input("direction (1 or -1):"))

def offsetPoint(point, unoffset=False): #offset point to center
        if(unoffset): return (point[0] + offsetPoint.offset[0], point[1] +
offsetPoint.offset[1])
        else: return (point[0] - offsetPoint.offset[0], point[1] - offsetPoint.offset[1])


def loadImage(infilename):
   img = Image.open(infilename)
   data = np.array(img)
   return data

def saveImage(npdata, outfilename):
   img = Image.fromarray(npdata)
   img.save(outfilename)


def getSwirlDegree(point,radius): #get degrees to rotate per point
        px,py = point
        dist = round(math.sqrt(px*px + py*py))
        #print(dist)
        if(dist>radius):
                return -1
        else:
                return 360.0 * getSwirlDegree.numswirls * ((radius - dist)/radius)

getSwirlDegree.numswirls = int(input("Max swirl turns:"))

#Set up input file params
im = loadImage(inf)
imagex = len(im)
imagey = len(im[0])
_offset= (int(round(imagex/2)), int(round(imagey/2)))
offsetPoint.offset=_offset

#copy array as result (same size)
result = np.array(im)

#print dimensions
print(result.shape)

a,b,c = result.shape

#empty result array
for i in range(0,a):
        for j in range(0,b):
                for k in range(0,c):
                        result[i][j][k]=0


#get largest contained circle radius
radius = 0
if(imagex>imagey): radius=int(round(imagey/2))
else: radius = round(imagex/2)

#How much maximum rotation degree will be



for i in range(0,imagex):
        for j in range(0,imagey):
                realPoint= offsetPoint((i,j))
                newCoords = rotateCoo(realPoint,getSwirlDegree(realPoint,radius))
                if(newCoords[0] == -1 and newCoords[1] == -1):
                        result[i,j,0] = im[i][j][0]
                        result[i,j,1] = im[i][j][1]
                result[i,j,2] = im[i][j][2]
                continue
        newCoords = offsetPoint(newCoords,True)
        #print(newCoords)
        #print(result[newCoords[0]][newCoords[1]])
        _x, _y = newCoords
        result[_x,_y,0] = im[i][j][0]
        result[_x,_y,1] = im[i][j][1]
        result[_x,_y,2] = im[i][j][2]
        #print(result[_x-1,_y-1,0])

#Blur missing points in result image, lost to floating error.
'''for i in range(0,a-1):
    for j in range(0,b-1):
        for k in range(0,c):
            if(result[i][j][k]==0):
                #print("Found floating precision error. Blurring")

result[i,j,k]=(int(result[i+1,j,k])+int(result[i-1,j,k])+int(result[i,j+1,k])+int(result[i,j-1,
k]))/4
'''
saveImage(result,outf)
```

Both with and without blur fixing:


TMHC{Th1sIsYOu1sntIt}


TMHC{Th1sIsYOu1sntIt}