Call for papers:

Exiftool reveals “SCIpher” as creator

web search for it returned a website:

https://pdos.csail.mit.edu/archive/scigen/scipher.html

which decrypts the document to the following:

https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/5e171aa074f390965a12fdc240.png

I liked this, as I imagine will everyone who’s ever had to wade through scientific literature in their time :-D

Just watch:

It was as simple as getting a diagram of sign language, and then as simple as not getting ‘g’ and ‘z’ mixed up to get the flag. :-)

givemeasign

Mirror:

The supplied file type is just data, but it’s definitely ‘archive.zip’ spelled backwards, a quick look with a hex editor confirms the bytes for a zip file are in reverse order. I tried a few services or tools to change endianness, but it proved annoying, so I decided to write a short python script to do what I could see needed to be done:

```
from array import array
a=array('B',open('evihcra.piz','rb').read())
new=open('ARCHive','wb')
d = a[::-1]
new.write(d.tostring())
new.close()
```

This gets the archive. Unzipping it gives another backwards filename of type “data”. The hex editor reveals that everything seems to be in order apart from the suspicious GNP header… switching it to PNG gives a .png file… it’s a mirror image of an egg with a qr code. Now, qr codes I’ve noticed, scan when people stick them on upside down, this one scans when apparently a mirror image of itself, and I get the egg.

Shell we argument:

I liked this one, because I only needed to get as far as decoding the command “eval” when reversing it… My next stroke of genius, which I swear was all on purpose (ahem) was to simply omit the space between eval and the list of variables, this had the happy result of turning the whole damn thing inside out, dumping out all of the strings and all of the things. The best of these were as follows:

```
echo "Find your egg at $t"
```

```
t="$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Ez$Fz$Kz$Lz$Mz$Nz$Oz$Pz$Ez$Fz$Gz$Hz$Iz$Qz$
Rz$Sz$Tz$Uz$Vz$Wz$Xz$Yz$Zz$az$bz$cz$dz$ez$fz$gz$hz$iz$jz$kz$lz$mz$nz$oz$Zz$pz$qz$
rz"
```

and finally more variables which, when appended to the list in the script allowed my cunning eval of $t (which had spat out gibberish) to spit out the following URL amidst the tortured screams of the mangled script next time it ran

https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e156242c74af38c.png

The script was later lovingly reassembled out of respect.

Sloppy encryption:

Tried doing it in python but the ways I tried didn't seem equivalent, so I learned a bit of ruby, essentially reversing the little script and keeping the gets function because it gives it character:

```
require"base64"
puts "enter b64 to mangle: "
b=gets.chomp
f=Base64.decode64(b)
g=f.scan(/./).map(&:ord).map{|x| '%02x'%x}.join
k=g.to_i(16)/['5'].cycle(101).to_a.join.to_i
l=k.to_s(16).scan(/../).map(&:hex).map(&:chr).join
puts l
```

Twisted:

I hated this one. Really hated it. I untwisted the qr to a point I could read it, even if a qr scanner couldn't, then I really just wrote the bits as string a couple of rows at a time whenever I thought of it. Then I wrote python to turn a string into a qr code, which is a trick I've used before when half-assing things:

```
from PIL import Image
fi=open('bits','r')
li=fi.readlines()
bi=[c.strip('\n') for c in li]

im=Image.open('blank.png')
pix=im.load()

for a in range(len(bi)):
  for b in range(len(bi[a])):
    if bi[a][b]=='1':
      pix[a+1,b+1]=(0,0,0)
    if bi[a][b]=='0':
      pix[a+1,b+1]=(255,255,255)
```

```
im.save('qr.png')
im.close()
```

Good fun this year, but sadly didn't have enough time to spend on good old Hacky Easter. Sad times.. Roll on hackvent :-D