P (38 solves, 137 pts)

Deskripsi Soal

```
P P P

Attachment : p.pcap
Author     : Bayu Federa
```

Diberikan *attachment* berupa *packet data file* bernama **p.pcap**. Sebelum melakukan analisis lebih lanjut, Kita lakukan beberapa enumerasi informasi. Adapun proses ini dapat dilakukan dengan bantuan **tshark**.

Hierarchical analysis

Sebagai acuan pengerjaan, kita lakukan enumerasi terhadap struktur dari paket data. Hasilnya dapat diketahui bahwa terdapat **843 packet.frame** yang memuat **ICMP Layer**. Dari sini kita dapat berasumsi terlebih dahulu bahwa mungkin saja terdapat informasi yang dimuat ke dalam **ICMP.data** sebagaimana problem ctf pada umumnya.

```
$ tshark -r p.pcap -q -z io,phs

===================================================================
Protocol Hierarchy Statistics
Filter:

sll                                      frames:843 bytes:84340
  ip                                     frames:843 bytes:84340
   icmp                                  frames:843 bytes:84340
===================================================================
```

Packet conversation analysis

Setelah mengetahui struktur dari paket data, kita lakukan enumerasi terhadap *network flow* dari komunikasi antara **client** dan **server**. Hal ini bertujuan agar nantinya proses eksfiltrasi data menghasilkan informasi yang relevan. Hasilnya kita mengetahui

bahwa terdapat tiga buah **client (13.78.65.168, 182.1.70.81, 80.82.65.234)** yang melakukan **ICMP Requests** pada sebuah **server (206.189.84.162)** dengan frekuensi *request* yang berbeda-beda. Dari sini kita bisa mengeliminasi **client 80.82.65.234** dalam proses analisis megingat kecilnya frekuensi request yang berlangsung.

```
$ tshark -r p.pcap -q -z conv,ip
================================================================================
IPv4 Conversations
Filter:<No Filter>
                                               |       <-      | |       ->
                                               | Frames  Bytes | | Frames  Bytes
13.78.65.168        <-> 206.189.84.162           368     36800     368     36800
182.1.70.81         <-> 206.189.84.162           53      5300      53      5300
80.82.65.234        <-> 206.189.84.162           1       140       0       0
================================================================================
```

Packet data overview

Adapun setelah menelaah beberapa informasi sebelumnya, berikut ini merupakan *snippet* dari paket data.

```
$ tshark -r p.pcap | head -2
1   0.000000 13.78.65.168 → 206.189.84.162 ICMP 100 Echo (ping)
    request id=0x2e26, seq=1/256, ttl=51

2   0.000049 206.189.84.162 → 13.78.65.168 ICMP 100 Echo (ping)
    reply id=0x2e26, seq=1/256, ttl=64 (request in 1)
```

Packet Filtering

Berbekal pemahaman yang diperoleh, kali ini kita lakukan proses filtering pada **ICMP Request** pada host **13.78.65.168** dan **182.1.70.81** dengan *constraint* pencarian pada **ICMP.data** yang umumnya dapat ditemukan pada offset **0x34 (52)**.

```
$ tshark -r p.pcap -Y 'ip.src eq 13.78.65.168' -x | grep 0030 | head -5
0030  00 00 00 00 ab 17 0a 00 00 00 00 00 10 11 12 13   ................
0030  00 00 00 00 84 1f 0a 00 00 00 00 00 10 11 12 13   ................
0030  00 00 00 00 a5 25 0a 00 00 00 00 00 10 11 12 13   .....%..........
0030  00 00 00 00 23 2b 0a 00 00 00 00 00 10 11 12 13   ....#+..........
0030  00 00 00 00 f5 30 0a 00 00 00 00 00 10 11 12 13   .....0..........
```

```
$ tshark -r p.pcap -Y 'ip.src eq 182.1.70.81' -x | grep 0030 | head -5
0030  00 00 00 00 65 a4 09 00 00 00 00 00 56 41 42 43   ....e.......VABC
0030  00 00 00 00 db 9f 0b 00 00 00 00 00 55 41 42 43   ............UABC
0030  00 00 00 00 bf 34 0d 00 00 00 00 00 35 41 42 43   .....4......5ABC
0030  00 00 00 00 d5 32 0f 00 00 00 00 00 4a 41 42 43   .....2......JABC
0030  00 00 00 00 0e 76 01 00 00 00 00 00 56 41 42 43   .....v......VABC
```

Dari sini, terlihat bahwa pada host **13.78.65.168,** yang memiliki **ICMP Request** paling banyak, tidak ditemukan informasi yang menarik untuk digali lebih lanjut. Sedangkan, pada host **182.1.70.81** terdapat *string sequence* yang kemungkinan *satisfiable* dengan charset *Base64* yang memuat isi dari flag yang dicari.

Data Exfiltration

Berdasarkan asumsi sebelumnya, kita mengetahui bahwa terdapat *sequence Base64 string* pada *range byte* ke 0x8 (8). Untuk membuktikan asumsi tersebut, kita lakukan proses eksfiltrasi data sebagai berikut

```
$ tshark -r p.pcap -Y 'ip.src eq 182.1.70.81' -Tfields -e data | cut -c17-18 | xxd -r -p
VU5JVFkyMDIwe1BJTkdfUE9OR19TZWlrYWlfRGVzdSEhISE6RH0=
```

Sebagai penjelasan, perintah tersebut akan melakukan filtering pada setiap paket yang berasal dari **182.1.70.81** barulah kemudian akan diambil fields **ICMP.data**. Selanjutnya, kita lakukan proses trimming untuk mendapatkan substring pada indeks ke 16 dan 17. Barulah dilakukan *hex decoding* dan *Base64 decoding* untuk mendapatkan flag yang dicari.

```
$ base64 -d <<< VU5JVFkyMDIwe1BJTkdfUE9OR19TZWlrYWlfRGVzdSEhISE6RH0=
UNITY2020{PING_PONG_Seikai_Desu!!!!:D}
```

Flag : UNITY2020{PING_PONG_Seikai_Desu!!!!:D}

NEET Diary (0 solves, 500 pts)

Deskripsi Soal

```
Hai hai Kazuma desu.

Last year, I was reincarnated into the fantasy world after being hit by
Truck-kun. Moreover, I was entrapped with a useless goddess, a chuunibyou and
weird companion. Overall, this is a diary of my own journey

Attachment : art-book.pdf
Author     : コト
```

Diberikan *attachment* berupa *PDF file* bernama **art-book.pdf**. Sebelum melakukan analisis lebih lanjut, Kita lakukan beberapa enumerasi informasi sedemikian sehingga kita dapat menemukan anomali yang ada pada *file*. Dari sini dapat dirincikan beberapa langkah dan opsi pengerjaan sebagai berikut:

Identify PDF Metadata

Sebelum kita beranjak lebih jauh, ada baiknya bagi kita untuk sejenak mengulik informasi *metadata* dari sebuah dokumen. Adapun proses ini dapat dilakukan dengan bantuan **exiftool**.

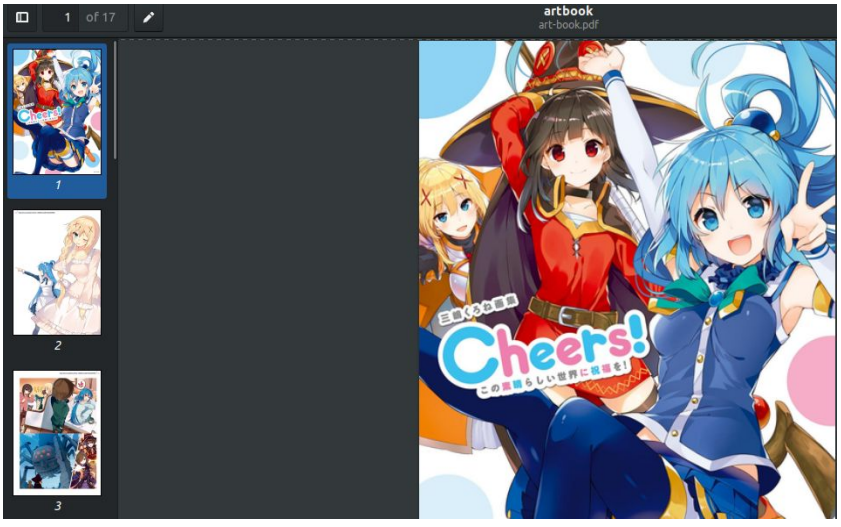
```
$ exiftool art-book.pdf
ExifTool Version Number         : 11.10
File Name                       : art-book.pdf
Directory                       : .
File Size                       : 7.9 MB
File Modification Date/Time     : 2020:02:27 08:02:55+07:00
File Access Date/Time           : 2020:03:15 16:01:09+07:00
File Inode Change Date/Time     : 2020:02:27 08:04:21+07:00
File Permissions                : rw-r--r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.3
Linearized                      : No
Modify Date                     : 2020:02:27 00:49:24
Create Date                     : 2020:02:27 00:49:24
Title                           : artbook
Producer                        : https://www.imagemagick.org
Page Count                      : 17
```

Terdapat beberapa informasi yang dapat kita aplikasikan nantinya, salah satunya diketahui bahwa versi PDF yang digunakan adalah **PDF 1.3** dimana *cross-reference table* masih terbilang sederhana sehingga penggunaan *object reference* dan *stream object* tidak sebanyak pada versi **PDF >= 1.4**.

Informasi selanjutnya yang kita peroleh adalah PDF tersebut didasarkan dari proses konversi sekumpulan *Image* ke PDF dengan **ImageMagick module**. Perlu diketahui bahwa sebelum melakukan konversi Image ke dalam PDF, **ImageMagick** akan melakukan identifikasi terhadap **Image MIME** terlebih dahulu dengan tujuan untuk menentukan jenis kompresi yang akan digunakan. Dari sini akan terbentuk **PDF object** yang memuat *stream* object **/Resources**. Objek ini memiliki *dictionary value* yang memuat referensi **/XObject** yang memuat binary dari Image dan **/Contents** yang memuat konfigurasi kanvas dari *Image*. Selain itu, akan terbentuk juga object **/Thumb** yang menjadi *thumbnail* dari **/XObject**.

Sebagai contoh, ketika suatu citra **JPEG** dikonversi ke dalam PDF, maka *stream object* akan memiliki properti **/Filter /DCTDecode** yang memuat **/XObject** berupa RAW data dari JPEG itu sendiri. Lain halnya dengan PNG, *stream object* akan memiliki properti **/Filter /FlateDecode** yang memuat **/Xobject** berupa **Zlib data** yang berasal dari **IDAT chunk.** Hal ini menyebabkan proses file carving menjadi sulit dilakukan karena dibutuhkan beberapa properti tambahan untuk dapat mengekstrak **PNG Image.**

Terakhir adalah dokumen PDF memuat 17 halaman berisikan **PNG Image** dari official artbook Konosuba yang merupakan karya dari ilustrator 三嶋くろね (@mishima_kurone). Lebih lengkapnya dapat disimak di laman berikut ini https://imgur.com/r/LightNovels/Ftz4i :D



Understanding PDF in a short way

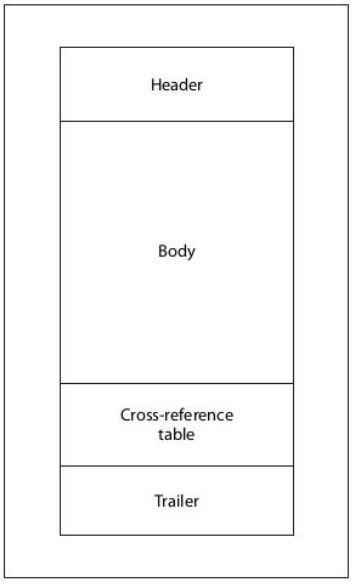
Konon terdapat pepatah lama yang berbunyi *Tak kenal maka tak sayang* sehingga untuk dapat memahami sesuatu lebih jauh, maka kita dituntut untuk mengenalnya lebih dekat. Pada *section* ini, akan sedikit dibahas terkait dengan beberapa aspek dasar dari **PDF File**. Lebih lengkapnya dapat dikaji lebih lanjut pada dokumentasi resmi PDF.

A. PDF Object

   Merupakan komponen penyusun dari PDF yang mendefinisikan syntax dan properti. Terbagi dalam 8 tipe objek dasar di antaranya: Boolean, Integer and Real numbers, Strings, Names, Arrays, Dictionaries, Streams, dan objek null.

B. File Structure

   Merupakan komponen yang menentukan pengolahan proses pada PDF, seperti halnya bagaimana objek disimpan, diakses maupun diperbaharui. Secara hirarkis, struktur **PDF File** dapat diilustrasikan sebagai berikut:



   Dari ilustrasi tersebut dapat diketahui bahwa terdapat 4 buah komponen minimum penyusun dari **PDF File**, yakni:

   1) *Header* yang memuat strings %PDF-{n-version}
   2) *Body* yang memuat sekuens dari **/Indirect Object** yang merepresentasikan isi dari dokumen.
   3) *Cross-Reference Table* yang dapat diilustrasikan sebagai katalog untuk setiap object yang terdapat pada PDF. Komponen ini diperlukan dalam pengaksesan objek secara instan tanpa membaca keseluruhan isi file. Oleh karenanya, komponen ini memuat isi dari semua offset tempat semua object berada.

4) *Trailer* yang menandakan *end-of-file* dari **PDF File**. Adapun pada komponen ini terdapat deklarasi jumlah total PDF bytes dimulai dari *header* hingga *xref table*. Selain itu terdapat pula properti **/Size** yang menunjukkan total minimum objek yang dibutuhkan oleh *xref table*

C. Content streams
Merupakan komponen yang memuat sekuens instruksi yang mendefinisikan bagaimana rupa dari halaman PDF atau entitas grafis lainnya

PDF Static Analysis

Setelah cukup mengetahui konsep tersebut, saat nya dilakukan analisis terhadap struktur **PDF File** yang diberikan. Dari sini kita akan mengkaji tentang objek apa saja yang dimuat dan adakah anomali pada struktur PDF yang diberikan. Adapun pada tahap ini digunakan **pdf-parser** untuk pemrosesan dokumen PDF yang ada.

```
$ pdf-parser.py --stats art-book.pdf
Comment: 2
XREF: 1
Trailer: 1
StartXref: 1
Indirect object: 181
  107: 20, 22, 23, 24, 25, 27, 28, 29, 30, 32, 33, 34, 35, 37, 38, 39, 40, 42,
43, 44, 45, 47, 48, 49, 50, 52, 53, 54, 55, 57, 58, 59, 60, 62, 63, 64, 65, 67,
68, 69, 70, 72, 73, 74, 75, 77, 78, 79, 80, 82, 83, 84, 85, 87, 88, 89, 90, 92,
93, 94, 95, 97, 98, 99, 100, 102, 103, 104, 105, 107, 110, 111, 112, 114, 116,
118, 119, 121, 122, 123, 124, 126, 130, 133, 134, 142, 143, 148, 149, 150, 151,
152, 153, 155, 157, 158, 161, 162, 165, 169, 172, 173, 174, 175, 176, 178, 181
 /Catalog 1: 1
 /Page 17: 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
 /Pages 1: 2
 /XObject 55: 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, 96,
101, 106, 108, 109, 113, 115, 117, 120, 125, 127, 128, 129, 131, 132, 135, 136,
137, 138, 139, 140, 141, 144, 145, 146, 147, 154, 156, 159, 160, 163, 164, 166,
167, 168, 170, 171, 177, 179, 180
```

Hasil statistik tersebut menunjukkan bahwa terdapat **181 object** yang masing-masing terbagi ke dalam objek:

1. */Catalog* yang memuat *object reference* dari objek */Pages*
2. */Pages* yang memuat instance dari keseluruhan dari *object reference /Page*
3. */Page yang memuat* isi dari sebuah halaman
4. */XObject* yang memuat *content stream* dari objek citra grafis
5. */Indirect object* atau *labeled object* yang digunakan sebagai *object reference*

Dari sini kita dapat mengatakan bahwa **PDF File** valid secara struktural mengingat komponen di atas merupakan komponen minimum yang diperlukan oleh **PDF Reader** untuk memproses dan menampilkan isi dari dokumen.

Kendati demikian, kita belum dapat mengidentifikasi anomali dari **PDF File** yang diberikan. Dari sini timbul sebuah pertanyaan, apakah terdapat *dangle object* atau objek yang tidak digunakan sebagai *object reference*? Contohnya saja, andaikan objek tersebut tidak memiliki */Page object* yang umumnya merupakan wadah bagi *content stream*, apakah **PDF Reader** mampu memproses dan menampilkan konten tersebut? Tentu saja tidak, karena *object reference* diperlukan untuk menghubungkan objek satu dengan objek lainnya sehingga ketika terdapat *missing link* pada struktur PDF, maka objek tersebut secara teknis akan diabaikan.

Terdapat beberapa tools yang dapat digunakan untuk menguji kesesuaian *case* tersebut, salah satunya adalah *ghostscript*. Dari sini kita akan melakukan *try-catch* untuk *stderr* yang dikeluarkan oleh *ghostscript* selama proses recover berlangsung. Adapun hasilnya ialah sebagai berikut:

```
$ gs -o repaired.pdf -sDEVICE=pdfwrite art-book.pdf
**** Warning: Discovered more enties in xref than declared in trailer /Size
Processing pages 1 through 17.
Page 1
Page 2
..
..
Page 16
Page 17
**** This file had errors that were repaired or ignored.
```

Yup, terdapat salah satu penanda *stderr* pada **PDF Trailer** dimana ditemukan entitas berlebih pada *xref table*. Untuk menguji deduksi tersebut, kita lakukan pengecekan terhadap *Trailer* **/Size**.

```
$ grep -a '/Size' art-book.pdf
/Size 106
```

Dari pemahaman tersebut, kita dapat menyimpulkan bahwa terdapat kontradiksi antara **Trailer Size** dengan **Xref table** dimana xref table memiliki total **182 object** lebih banyak daripada trailer size yang seharusnya seragam dari segi jumlah. Oleh karena itu, kita dapat membuktikan asumsi bahwa terdapat **76 dangle object** yang

ditambahkan pada **PDF File** yang kemungkinan memuat informasi dari flag yang dicari. Berbekal pemahaman itulah petualangan baru pun dimulai :)

Digging out the Dangle Object

Seperti kita ketahui sebelumnya, bahwa **PDF File** yang asli hanya memiliki **105 object + 1 zero object**, sehingga kita dapat memisahkan semua objek dimulai dari objek ke **106**. Hal ini dimaksudkan agar proses analisis dan *file parsing* menjadi lebih mudah untuk dilakukan.

```
$ grep -a '106 0 obj' -A 9999 art-book.pdf > trim.pdf
```

Kemudian kita lakukan kembali pengecekan terhadap statistik **PDF File**

```
$ pdf-parser.py --stats trim.pdf

Comment: 1
XREF: 1
Trailer: 1
StartXref: 1
Indirect object: 76
  38: 107, 110, 111, 112, 114, 116, 118, 119, 121, 122, 123, 124, 126, 130, 133,
134, 142, 143, 148, 149, 150, 151, 152, 153, 155, 157, 158, 161, 162, 165, 169,
172, 173, 174, 175, 176, 178, 181
 /XObject 38: 106, 108, 109, 113, 115, 117, 120, 125, 127, 128, 129, 131, 132,
135, 136, 137, 138, 139, 140, 141, 144, 145, 146, 147, 154, 156, 159, 160, 163,
164, 166, 167, 168, 170, 171, 177, 179, 180
```

Hasilnya dapat kita ketahui bahwa terdapat sisa **/XObject** sebanyak 38 object sehingga dapat dipastikan flag tersimpan dalam representasi citra yang kemungkinan besar di generate oleh **ImageMagick**. Adapun sisa 38 object lainnya kemungkinan memuat properti **/Contents** dari masing-masing entitas **/XObject.**

Extract Image from Corresponding /XObject

Setelah mendapatkan secercah pencerahan, akhirnya kita bertemu pada kesimpulan bahwa kita perlu merekonstruksi citra dari referensi **/XObject** yang tersedia. Namun sebelum itu kita perlu mengidentifikasi jenis kompresi apa yang digunakan sebagai acuan penentuan jenis citra.

```
$ grep -a '/Filter' trim.pdf | uniq
/Filter [/ASCII85Decode /FlateDecode]
```

Hasilnya, dapat diketahui bahwa citra merupakan **PNG File** yang di wrap dengan ASCII85 encoding. Hal ini menandakan bahwa kita cukup melakukan **ASCII85 decode** dan **Zlib decompress** sebelum akhirnya kita dapat merekonstruksi citra yang ada.

Kendati demikian, sebelumnya kita perlu melakukan pengecekan terhadap parameter lainnya, seperti halnya sekuens **/Name**, sehingga informasi yang dihasilkan dapat dipastikan berada pada *order* yang sesuai

```
$ grep -oP '/Name /F\d+' trim.pdf | head
/Name /F28
/Name /F23
/Name /F33
/Name /F20
/Name /F29
/Name /F34
/Name /F24
/Name /F26
/Name /F9
/Name /F3
```

Yup, ternyata kita masih perlu melakukan *re-plotting* pada bagian akhir operasi sedemikian sehingga citra yang dihasilkan menjadi terurut.

Final Phase !

Setelah mendapatkan informasi yang cukup terkait dengan *constraint* yang ada, kita dapat memulai untuk mengekstraksi citra. Terdapat beberapa opsi yang dapat penulis berikan, di antaranya:

I. Utilize pdf-parser *--filter* feature

Pada opsi ini kita akan menggunakan beberapa fitur yang disediakan oleh **pdf-parser**, salah satunya adalah argumen *--filter* yang difungsikan untuk mengekstraksi raw data dari kombinasi encoding maupun kompresi sebagaimana tertera pada detil *usage*.

```
  -f, --filter            pass stream object through filters
(FlateDecode,
                          ASCIIHexDecode, ASCII85Decode, LZWDecode and
                          RunLengthDecode only)
```

Dari sini kita lakukan listing untuk mendapatkan *object.id* dari masing-masing **/XObject**.

```
$ pdf-parser.py --stats trim.pdf | grep -oP '(?<=/XObject \d{2}:).*' | tr
', ' '\n' | tr -s '\n' | grep . > xobject

$ head xobject
106
108
109
113
115
117
120
125
127
```

Barulah kemudian dilakukan proses *data dump* dengan bantuan **pdf-parser** sekaligus men-*catch key* **/Name**, **/Width**, dan **/Height**

```
$ xargs -a xobject -Iz bash -c "pdf-parser.py trim.pdf -o z -f -d
stream/z.bin | grep -oP '(?<=obj )\d+|(?<=Name /F)\d+|(?<=Width
)\d+|(?<=Height )\d+' | xargs" >> data

$ head -5 data
106 28 600 490
108 23 535 490
109 33 605 380
113 20 515 270
115 29 605 380
```

Terakhir dilakukan proses **Image builder** dengan bantuan **module Pillow.**

```
#!/usr/bin/python
# solve1.py
from PIL import Image

objs = open('data').read().split('\n')
for obj in objs[:-1]:
    imMode  = 'RGB'
    objId   = obj.split()[0]
    objName = obj.split()[1]
    wid,hei = map(int, obj.split()[-2:])
    data    = open('stream/%s.bin'%(objId), 'rb').read()
    Image.frombytes(imMode, (wid, hei),
data).save('img/%s.png'%(objName))
```

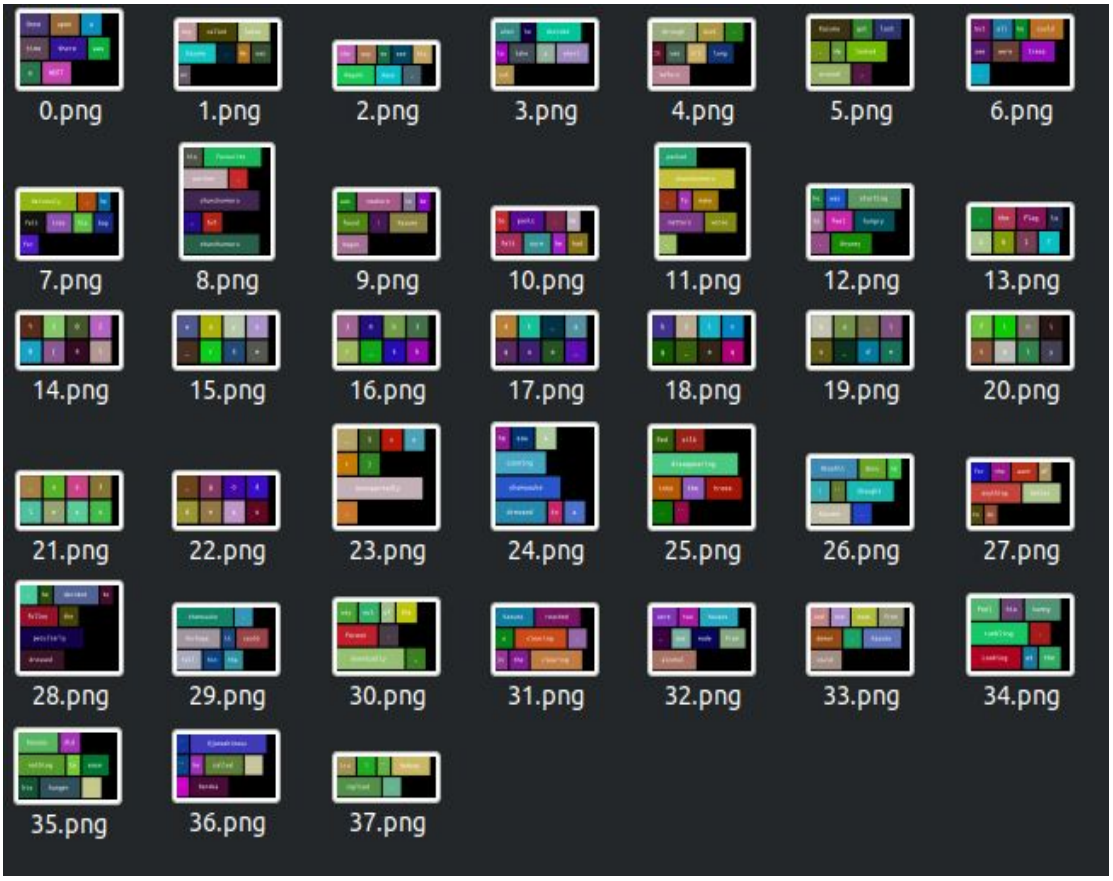
II. **Parse content stream using Python RegEx**

Hampir sama dengan opsi pertama, disini kita perlu mendefinisikan RegEx *rule* untuk mengekstrak *grouping* dari **/Name, /Width**, dan **/Height**. Adapun yang menjadi perbedaannya ialah disini kita perlu memproses *content stream* dengan operasi **a85decode** & **zlib.decompress** secara berturut-turut. Berikut implementasinya:

```
#!/usr/bin/python
# solve1.py
from PIL import Image
from base64 import a85decode
import re, zlib

file = open('trim.pdf','rb').read()
objs = re.compile(b'\d+ \d+ obj(.*?)endobj \n*', re.S)
rule = re.compile(b'/Name /F(\d+).*Width (\d+).*Height
(\d+).*stream\n(.*?)\nendstream', re.S)

for obj in objs.findall(file):
    matches = rule.findall(obj)
    if matches:
        matches = matches[0]
        objName = matches[0].decode()
        wid,hei = map(int, matches[1:3])
        data  = zlib.decompress(a85decode(matches[-1]))
        Image.frombytes('RGB',(wid,hei),
data).save('img/%s.png'%(objName))
```

Adapun hasilnya diperoleh *Image sequence* sebagai berikut

```
$ python solve.py
```




, the flag is Y 2 0 2
U N I T 0 { 4 l
w a y s 3 m b 3
_ r 3 m r _ t h
4 t _ a b 3 i n
q u a _ g _ a q
u a _ i f i n i
s _ d e t e l y
_ u s 3 _ g o d
l e s s d e s s
_ 3 v e
r }

** Alternative

Kendati demikian terdapat opsi pengerjaan lain yang dapat digunakan untuk memperoleh flag yang diminta, salah satunya ialah dengan menambahkan *object* **/Page** untuk setiap **/XObject** yang tersedia , barulah kemudian dilakukan penambahan *object reference* pada *object* **/Pages**. Terakhir dilakukan perhitungan ulang pada ***xref table*** dan **PDF Trailer**.

Adapun berikut merupakan implementasi beserta hasil pemrosesan dari skema di atas

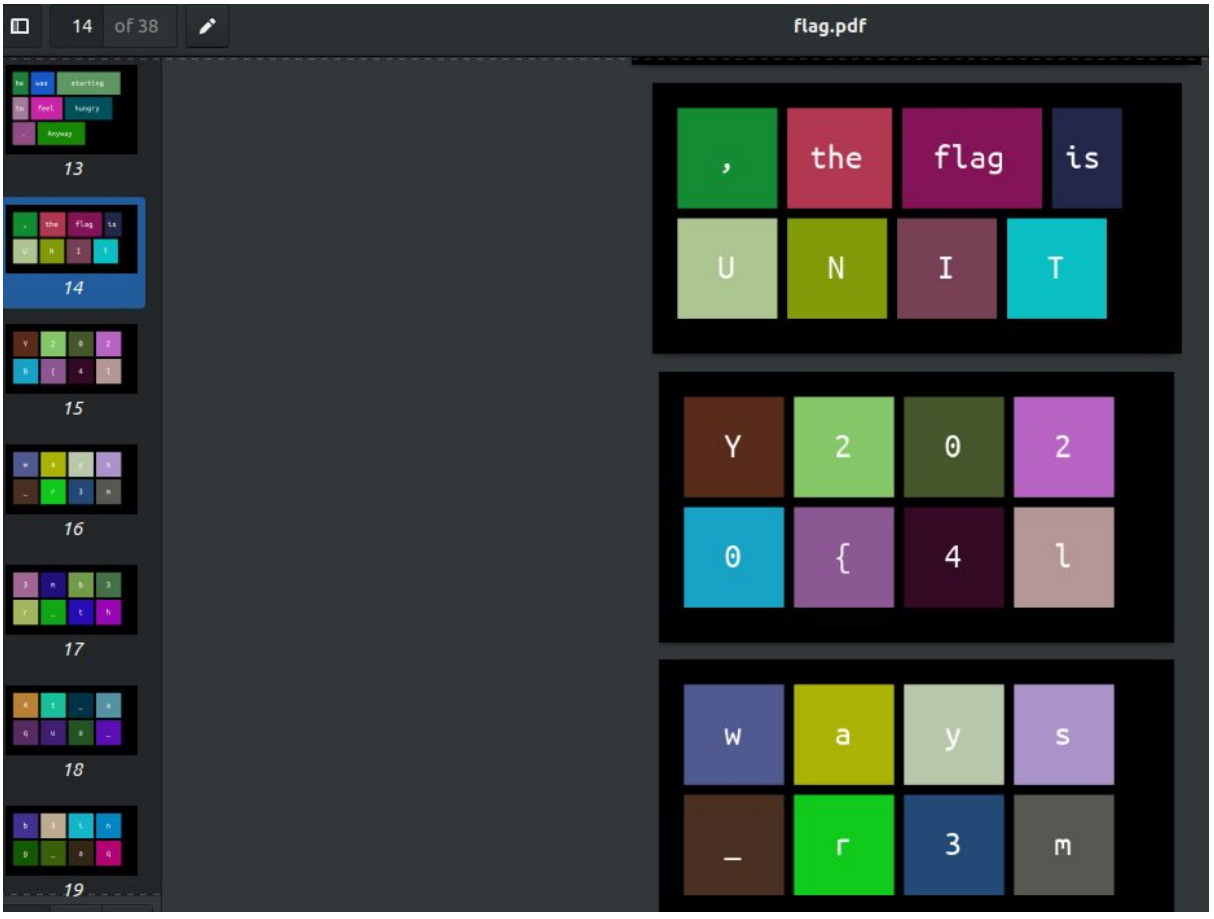
```
#!/usr/bin/python
# solve2.py
import re

class PDFParser(object):
    def __init__(self, name, count=0):
        self.pdf   = open(name, 'rb').read()
        self.data  = dict()
        self.refer = dict()
        self.count = count
        self.parse()

    def parse(self):
        # Extract PDF Header
        head = re.compile('(^%PDF-(1.\d)\s?\n).*\d+ 0 obj', re.S)
        self.header  = head.search(self.pdf).group(1)
        self.version = head.search(self.pdf).group(2)
        # Extract object
        for objId in self.extract_object():
              self.extract_data(objId)

    def extract_object(self):
        rule     = re.compile('(\d+)( \d+ obj)(.*?)(endobj \n*)', re.S)
        if not self.count:
              raws     = map(list, rule.findall(self.pdf))
        else:
              raws     = map(list, rule.findall(self.pdf))[:self.count]
        objId = map(lambda _ : int(_[0]), raws)
        self.object = dict(zip(objId,raws))
        return self.object

    def extract_data(self, id):
        data = self.object[id][2]
        var  = re.compile('(/\w+) (.*)', re.S)
        for section in data.split('\n'):
              if var.match(section):
                    keyName = var.search(section).group(1)
```

```
                    values  = var.search(section).group(2)
                    if re.match('(/\w+) \d+ 0 R', section):
                          varname = self.refer
                    else:
                          varname = self.data
                    element = varname.get(keyName, dict())
                    if not element:
                          varname[keyName] = element
                    subelement = element.get(values, list())
                    if not subelement:
                          varname[keyName][values] = subelement
                    if id not in subelement:
                          subelement.append(id)

    def reassemble(self):
        data  = self.header
        data += self.get_objects()
        data += self.get_xref() + self.get_trailer(data)
        self.pdf = data

    def add_object(self, data, num):
        if not self.object.get(num):
              self.object[num] = data

    def get_trailer(self, data=''):
        trail     = re.compile('trailer.*\n(/Info \d+ 0 R\n).*\n/(Size
\d+)\n.*\n(\d+)\n%%EOF\n', re.S)
        trailer = trail.search(self.pdf).group()
        xrefbyt = trail.search(self.pdf).group(2)
        pdfbyte = trail.search(self.pdf).group(3)
        pdfinfo = trail.search(self.pdf).group(1)
        datalen = str(len(data))
        trailer = trailer.replace(pdfinfo,'')
        trailer = trailer.replace(xrefbyt, 'Size %s'%(len(self.object)+1))
        return trailer.replace(pdfbyte, datalen)

    def get_xref(self):
        xref = ' xref\n0 %s\n0000000000 65535 f \n'%(len(self.object)+1)
        pos = len(self.header)
        for obj,val in self.object.items():
              xref += '%010d 00000 n \n' % (pos)
              pos  += len(''.join(val))
        return xref

    def get_object_streams(self, _type=''):
        objStream = dict()
        for key, val in self.data['/Filter'].items():
              for obj in val:
                    if not _type or obj in self.data['/Type'][_type]:
                          data = self.object[obj][2]
```

```
                            stream = re.compile('stream(.*?)endstream',re.S)
                            objStream[obj] = stream.search(data).group(1)
        return objStream

    def get_objects(self):
        return ''.join(''.join(val) for obj,val
               in self.object.items())[:-1]

    def save(self, path):
        self.reassemble()
        with open(path, 'wb') as pd:
               pd.write(self.pdf)

def add_page(seq, name, img, obj, canvas):
    part     = str(seq)
    part += ' 0 obj'
    part += ' \n<<\n/Resources'
    part += ' \n<<\n/ProcSet [/PDF /Text /ImageC]'
    part += '\n/XObject \n<<\n{} {} 0 R'.format(name, obj)
    part += '\n>>\n>>\n/Contents {} 0 R'.format(img)
    part += '\n/Parent 2 0 R\n/Type /Page'
    part += '\n/MediaBox [0 0 {}]'.format(canvas)
    part += '\n/CropBox [0 0 {}]\n>>\n'.format(canvas)
    part += 'endobj \n'
    return part

base  = PDFParser('art-book.pdf', count=2)
pdf   = PDFParser('art-book.pdf')
xobj  = pdf.data['/Type']['/XObject']
pairs = {}
pg_li = []

for i,j in pdf.data['/Name'].items():
    if '/Im' not in i:
        pairs[i] = [pdf.data[i]['Do'][0],j[0]]

counter = 3
for i in range(len(pairs)):
    id  = '/F%s'%(i)
    img = pdf.object[pairs[id][0]]
    cnv = re.search('(\d+) 0 0 (\d+) 0 0 cm', img[2]).group(1,2)
    img[0] = str(counter+1)
    obj = pdf.object[pairs[id][1]]
    obj[0] = str(counter+2)
    pg_li += ['{} 0 R'.format(counter)]

    pages = base.object[2][2]
    kids  = re.search('\[.*\]', pages).group()
    count = re.search('/Count \d+', pages).group()
    pages = pages.replace(kids, '[{}]'.format(' '.join(pg_li)))
```

```
    pages = pages.replace(count, '/Count {}'.format(len(pg_li)))
    base.object[2][2] = pages

    page = add_page(counter,id, counter+1, counter+2,' '.join(cnv))
    base.add_object(page,counter)
    base.add_object(''.join(img),counter+1)
    base.add_object(''.join(obj),counter+2)
    counter += 3

base.save('flag.pdf')
```

```
$ python solve2.py
```



Flag: UNITY2020{4lways_r3m3mb3r_th4t_aqua_b3ing_aqua_is_definitely_us3less_goddess_3ver}