Write-up by BusyR

Intigriti is a crowdsourced security platform where security researchers and companies meet.

Table of Contents

FIND THE FLAG, EARN SWAG

Step 1: A tweet from @intigriti

It all starts with a simple tweet…

Since there are no links, no weird encodings or other hidden elements in the tweet, I guess the focus should start at the image.

Step 2: Analyzing the image

Download the image at https://pbs.twimg.com/media/DweADlgXgAAehHh.jpg:large and let `binwalk` have a look at it:

```
# binwalk -e DweADlgXgAAehHh.jpg\:large

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
```

```
0             0x0             JPEG image data, JFIF standard 1.01
182           0xB6            Zip archive data, at least v2.0 to extract, compressed size:
11029, uncompressed size: 12129, name: nottheflag.pdf
65660         0x1007C         End of Zip archive, footer length: 22
```

Nice, there's a hidden ZIP-file in the image, and binwalk was so nice to extract that ZIP for us.

```
-rw-r--r-- 1 root root 105240 Jan 16 16:07 B6.zip
-rw-r--r-- 1 root root  12129 Jan  8 16:53 nottheflag.pdf
```

Step 3: A zipped PDF

`nottheflag.pdf` was zipped inside `B6.zip`. `Binwalk` automaticly unzipped it for us, but what's weird is that the ZIP-file is so much bigger than the PDF.

A quick inspection using `xxd` learns that there's a really large blob of AAAA's inside the zip.

```
# xxd B6.zip
… cut ...
00002b50: 5bcb e120 6c56 5669 1519 ecff 0741 4141  [.. lVVi.....AA
00002b60: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002b70: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002b80: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002b90: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002ba0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002bb0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002bc0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002bd0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002be0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002bf0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00002c00: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
… cut …
# xxd B6.zip  | grep AAAAAAAAAAAAAAAA | wc -l
3392
# echo 16*3292 | bc
52672
```

However, that blob explains only about half of the size difference, let's ignore the other half for now…

The PDF contains 3 lines of text:

```
aHR0cHM6Ly9nby5pbnRpZ3JpdGkuY29tLzA3YjBmTDI0bGttdmE=
Source for this cool technique:
https://twitter.com/David3141593/status/1058124224798380032
```

The Tweet from @David3141593 explains a technique of embedding ZIP archives in Twitter-images, that survives the Twitter-thumbnailing-process. I guess that blob of AAAA's has something to do with that ;-)

The 1st line looks like a BASE64-encoded string:

```
# echo aHR0cHM6Ly9nby5pbnRpZ3JpdGkuY29tLzA3YjBmTDI0bGttdmE= | base64 -d
https://go.intigriti.com/07b0fL24lkmva
```

Step 4: A new zip-file

Following this BASE64-encoded URL leads to yet another zip-file: `data.zip`. However, this zip seems to be encrypted with a password:

```
# unzip data.zip
Archive:  data.zip
   creating: data/
[data.zip] data/1_177.jpg password:
```

Step 5: Trying to crack...

So, let's try to crack it... first convert to john using `zip2john`

```
# zip2john data.zip > data.zip.john
data.zip->data/ is not encrypted!
ver a  data.zip->data/ is not encrypted, or stored with non-handled compression type
ver 14  efh 5455  efh 7875  data.zip->data/1_177.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=96EE0CB5
ver 14  efh 5455  efh 7875  data.zip->data/1_163.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=96EE0CB5
ver 14  efh 5455  efh 7875  data.zip->data/1_188.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_349.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_70.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=201,

... cut ...

ver 14  efh 5455  efh 7875  data.zip->data/1_350.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_185.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_191.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_146.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=C79DD362
ver 14  efh 5455  efh 7875  data.zip->data/1_152.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=196,
decmplen=314, crc=96EE0CB5

# cat data.zip.john
data.zip:
$pkzip2$3*2*1*0*8*24*96ee*5700*99c036100884cbd6f0f9afa8db39bd2cce9df2b5a4796707f56624ea5164f373c
7bd188b*1*0*8*24*c79d*5700*ba0e0728b3b349c8df05bc7ebff3b3fe37c5587a7bd32351cfd4d2e83e09bd14c2e9c
1e5*2*0*c4*13a*96ee0cb5*3f*48*8*c4*96ee*5700*bcf72b7831e077f32d791779ecaae151fafd11ff23277f00a9a
fbac7a19503f9c339526b0cf4da55e499283558e2bcf89af32fa20037c4f70e3d24f98f6b86da4e8ec826ee2d7e26fa8
be41b72a3f6033a7373550ba819e3b534d2df6ce8c045f08a16f04245c222e8076c5eff8a3c83df3e3031d685a2465a7
e4f8701bf6870fb20846ce913f9f596ac722624a0017b0c618b2190b0cd7183ed23c3112818ab07abd2694a989d9670d
8152986cc2828ba4c06fc8e802d6a0d75a342ac50576cf086e9f5*$/pkzip2$:::::data.zip
```

And try to crack it using `rockyou.txt`, a nice wordlist that contains many real-life passwords:

```
# john data.zip.john --wordlist=rockyou.txt
Loaded 1 password hash (PKZIP [32/64])
guesses: 0  time: 0:00:00:07 DONE (Fri Jan 11 10:16:08 2019)  c/s: 1917K  trying:a6_123 -
*7Vamos!
```

No luck here. I've tried some other wordlists, and even let it bruteforce for a while, but that didn't seem to get us anywhere...

Step 6: Data analysis

Looking at the output from `zip2john`, a few things pop out… There's a LOT of jpg's in there, and that `zip2john` command we ran earlier has a lot of details on those files. Let's copy/paste that screen-output to a file called `zip2john.txt` and do some quick analysis to see what info we already have…

```
# cat zip2john.txt | wc -l
441
# cat zip2john.txt | cut -f3 -d= | cut -f1 -d, | sort | uniq -c
    390 314
      1 317
     50 318
# cat zip2john.txt | cut -f4 -d= | sort | uniq -c
     36 22EB0BB8
     14 5B808910
      1 81F9BF5A
    206 96EE0CB5
    184 C79DD362
# cat zip2john.txt | cut -f2 -d_ | cut -f1 -d\ | sort -n
01.jpg
02.jpg
03.jpg
04.jpg

... cut ...

438.jpg
439.jpg
440.jpg
441.jpg
```

So, based on the `decmplen` and the `crc`-values, we have just 5 different images, spread out over 441 very small jpeg's, that are between just 314 and 318 bytes each, and all the filenames are numbered sequential.

A quick test trying to create some small jpegs shows that just a single pixel jpeg can be done in 160 bytes, but that size depends a lot on the program being used. For example, using Gimp to create a single pixel jpeg resulted in a 538 bytes file.

```
# convert xc:[1x1\!] test_imagemagick.jpg
# ls -l test*.jpg
-rw-r--r-- 1 root root 538 Jan 16 16:54 test_gimp.jpg
-rw-r--r-- 1 root root 160 Jan 16 16:50 test_imagemagick.jpg
```

I guess it's safe to say that the images in the zip-file could not contain much more info than just a single pixel, and since we have the CRC-values of all the images, we can differentiate between them.

Step 7: Putting it all together

441 also happens to be 21^2, so let's try to create a 21x21 grid in LibreOffice Calc, and let's start coloring some cells ;-)

We don't know the original colors of the pixels, but it's safe to ignore that for now…

The initial result shows a familiar pattern:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
3	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
5	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
6	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126
7	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147
8	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168
9	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189
10	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
11	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231
12	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252
13	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273
14	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294
15	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315
16	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336
17	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357
18	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378
19	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
20	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420
21	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441
22																					
23																					

Ok, it seems we're trying to recreate a QR-code. Let's fix those colors and clean that image up a bit:

Cool. Now, just scan that image with a QR-Code app on your phone, , or upload it to some website that does online scanning, and we're presented with the flag:

flag:YOUWINTIGRITI