24.02.20

The Nixu Challenge (2020)

ACME Phone & Numb3rs

Introduction

This year there weren't as many challenges available as last year, but I still had fun solving them even though I enjoy web and pwn challenges more.

ACME Phone

Challenge description:

```
The customer (ACME) has delivered an iPhone backup that needs to be
investigated. They want to know the whereabouts of one of their
employees. It's broken, they said. It's impossible to access, they
said. Are you an elite forensics investigator? Can you rock?
```

Categories: Forensics, 2020

Introduction

The data given in the challenge is an iPhone backup, a password protected one, also more commonly known as an encrypted iPhone backup. For us this means that we can't extract data (Photos, contacts, appdata etc.) from the backup without knowing the password first.

Cracking The Password Using HashCat

The password "hash" is stored in a keychain inside the Manifest.plist.

The hash can be extracted in various ways, but I used a tool called itunes_backup2hashcat.pl from GitHub.

This backup was made on a iPhone running iOS 10 or above (13.2.3) the extracted data looks like this:

```
$itunes_backup$*10*b03fca19a6262c9157885ecb84d355985b56a32da288fc61f63e72cc020f34957e27a25a31bbdb83*10000*dfb440d2d35f4109fe35b654001c7011abedcfc1*10000000*588579d4fd71ef8518ee171d014429d4403531fa
```

Syntax of the hash: `$itunes_backup$*10*wkpy*iter*salt*dpic*dpsl`

Now let's crack it! There is a clue for us in the challenge description: `"Can you rock?"` This might refer to the famous rockyou.txt password list which is commonly used in dictionary and brute force attacks; it even comes stock with Kali Linux.

Assuming we already have HashCat installed, we first save the hash into a `.txt` file and then run the following command:

```
hashcat -m 14800 your_hash_file.txt /directory/to/rockyou.txt
```

`-m 14800` is letting HashCat know what type of hash it's trying to decrypt.

After a few minutes we get the password, which is “password”.

Extracting Data From The Backup

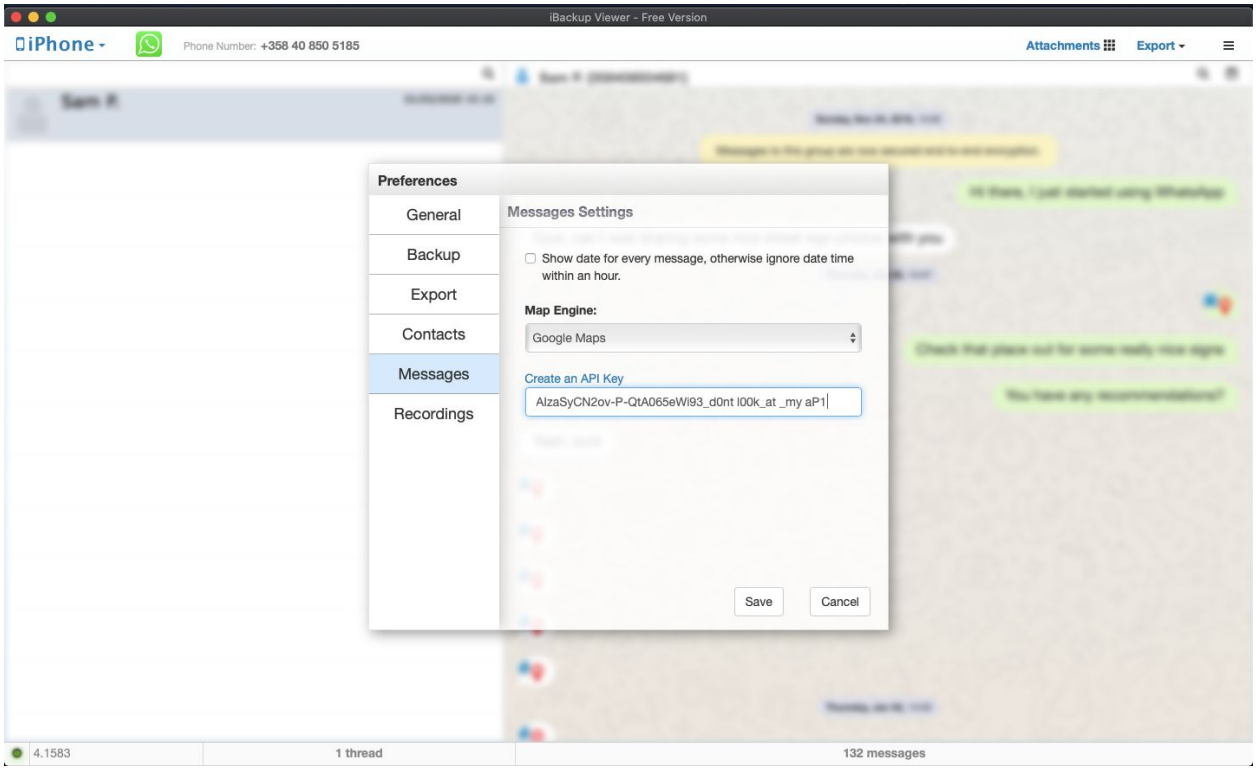
Now that the password has been retrieved, we need a tool to browse around the backup. Personally I used *iPhone Backup Extractor* and *iBackup Viewer* , but you can use a wide variety of different backup extractors.

The flag was scattered around the backup in 4 parts.

Part #1 (WhatsApp Attachments)

Looking around the backup in iBackup Viewer, I saw there were lots of location attachments in WhatsApp. Unfortunately I couldn't extract them without using the premium version of the application.

Looking around the preferences in iBackup Viewer, I found out I could use a Google API key to get the thumbnail for the attachments.



So I inserted my own Google API key and using WireShark, I managed to capture the data the app was receiving from Google as I refreshed the chat.

No.	Time	Source	Destination	Protocol	Lengtl	Info
57	0.740959	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37926,28.439052&z
84	1.143479	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37926,28.439152&z
111	2.109082	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37926,28.439352&z
159	2.708637	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37926,28.439752&z
549	8.187406	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.435152&z
420	6.156875	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.435552&z
460	6.760214	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.435852&z
470	6.947409	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.435952&z
489	7.293533	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.436352&z
181	2.986265	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.436752&z
212	3.396157	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.436952&z
260	3.975199	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.437352&z
283	4.358132	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.437752&z
306	4.660229	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.438152&z
352	5.179006	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.438552&z
18	0.205964	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.438752&z
90	1.207636	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.439152&z
116	2.176884	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.439352&z
164	2.779395	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37936,28.439752&z
375	5.452383	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.435152&z
425	6.246893	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.435552&z
475	7.025694	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.435952&z
494	7.378318	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.436352&z
186	3.074522	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.436752&z
218	3.458702	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.436952&z
266	4.043490	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.437352&z
300	4.583189	192.168.1.105	172.217.20.42	HTTP	371	GET /maps/api/staticmap?center=66.37946,28.437852&z

After saving the traffic into a pcap file, I exported the http objects from the pcap into a directory and parsed them by writing the following script in python.

```
import os, re

current_directory = os.getcwd()   # Get current working directory.
working_directory = current_directory+"\\images" # Change dir to “images”.

coordinate_regex = r"center=(.*?)\&" # Regex to get coordinates from name.

files = os.listdir(working_directory) # Save filenames in cwd into a list.

# Extract coordinates from filenames and save them to coordinates1.txt
with open("coordinates1.txt","w") as f:
    for item in files:
        f.write(re.findall(coordinate_regex, item)[0]+"\n")
```

Putting all the coordinations into a *GPS Point Plotter* gave us this:

Part #2 (Photos)

I extracted the photos one by one manually using iPhone Backup Extractor mainly because at the time I had no other choice.

After extracting all the photos into one folder I ran `exiftool * > exifdata.txt` to extract all the metadata from the images into a single `.txt` file that will be easy to parse.

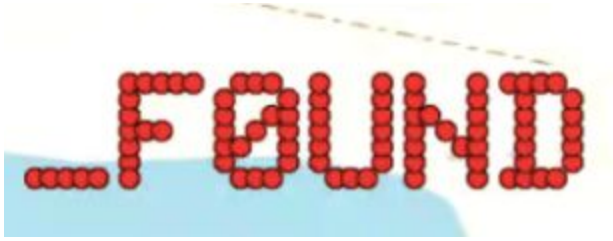
I parsed the `exifdata.txt` file with grep and putting the output into a file using the following command:

```
grep -r GPS Position > GPSPositions.txt
```

Unfortunately this gave us the coordinates in *Degrees, Minutes & Seconds (DMS)* format. However the GPS Point Plotter website requires the coordinates to be in *Decimal Degrees (DD)* format. We can easily convert the coordinates into DD format with a python script.

```
import re

# Function for reading the coordinates from a file.
def read_coordinates(file):
    with open(file) as coord:
        return list(coord.read().split("\n"))

# Function for dms to dd conversion.
def dms_to_dd(deg, min, sec, dire):
    dd = float(deg)+(float(min)/60)+(float(sec)/(60*60))
    if dire == 'W' or dire == 'S':
        dd *= -1
    return dd

# Function for parsing the coordinate data
def parse_dms(dms):
    coordinates = dms.split(",") # Split coordinates into parts (lat, lon)
    lat = coordinates[0]
    lon = coordinates[1]

    # Parse values with regex
    parts_lat = re.split('[deg\'"]+', lat.replace(" ",""))
    parts_lon = re.split('[deg\'"]+', lon.replace(" ",""))

    # Convert coordinates to DD
    lat = dms_to_dd(parts_lat[0], parts_lat[1], parts_lat[2], parts_lat[3])
    lon = dms_to_dd(parts_lon[0], parts_lon[1], parts_lon[2], parts_lon[3])

    return (lat,lon)

coordinates = read_coordinates("coordinates.txt")

with open("parsed.txt","w") as f:
    for item in coordinates:
        lat, lon = parse_dms(item)
        print str(lat)+', '+str(lon)
        f.write((str(lat)+', '+str(lon)+"\n"))
```

Now that we have the coordinates in the right format, we can toss them into the same GPS Point Plotter and get the second part of the flag.

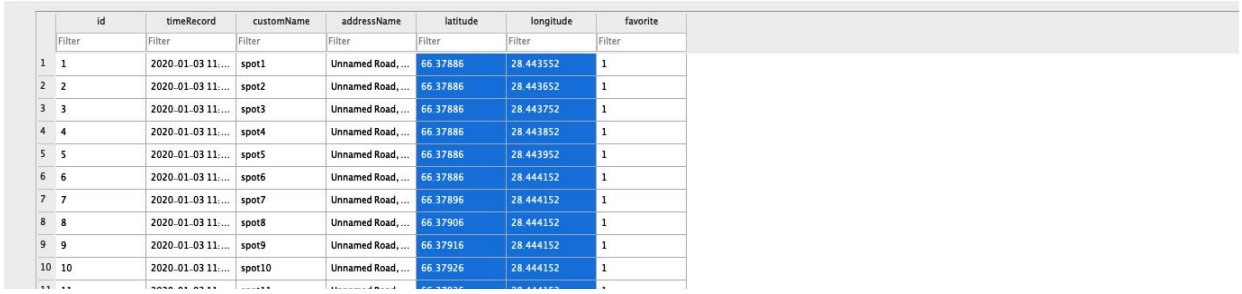


Part #3 (CompassMap)

Looking into the applications installed on the phone, we can see there's multiple applications that might be holding location data.

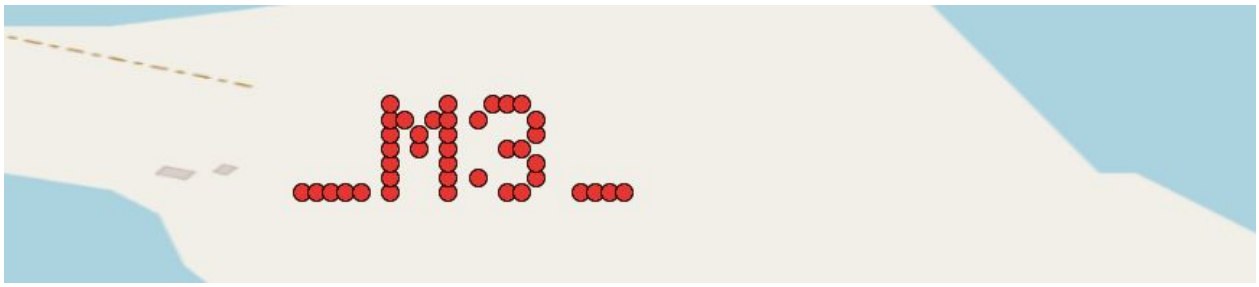
Using iPhone Backup Extractor I also extracted the database of the CompassMap application. (Applications > com.Luongbeta.CompassMap > Library > Application Support > db.sqlite)

This file is a sqlite database file and we can read it with [DB Browser for SQLite](). We can see more of the coordinates in the database.



	id	timeRecord	customName	addressName	latitude	longitude	favorite
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2020-01-03 11:...	spot1	Unnamed Road, ...	66.37886	28.443552	1
2	2	2020-01-03 11:...	spot2	Unnamed Road, ...	66.37886	28.443652	1
3	3	2020-01-03 11:...	spot3	Unnamed Road, ...	66.37886	28.443752	1
4	4	2020-01-03 11:...	spot4	Unnamed Road, ...	66.37886	28.443852	1
5	5	2020-01-03 11:...	spot5	Unnamed Road, ...	66.37886	28.443952	1
6	6	2020-01-03 11:...	spot6	Unnamed Road, ...	66.37886	28.444152	1
7	7	2020-01-03 11:...	spot7	Unnamed Road, ...	66.37896	28.444152	1
8	8	2020-01-03 11:...	spot8	Unnamed Road, ...	66.37906	28.444152	1
9	9	2020-01-03 11:...	spot9	Unnamed Road, ...	66.37916	28.444152	1
10	10	2020-01-03 11:...	spot10	Unnamed Road, ...	66.37926	28.444152	1

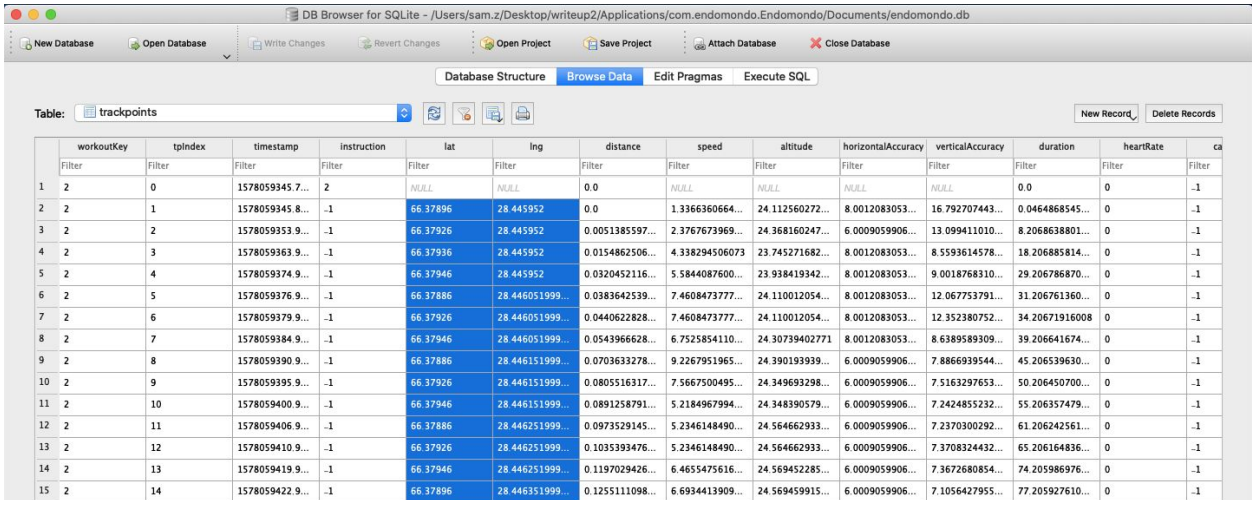
After copying the coordinates and tossing them into the GPS Point Plotter we get the 3rd part of the flag.



Part #4 (Endomondo Sports Tracker)

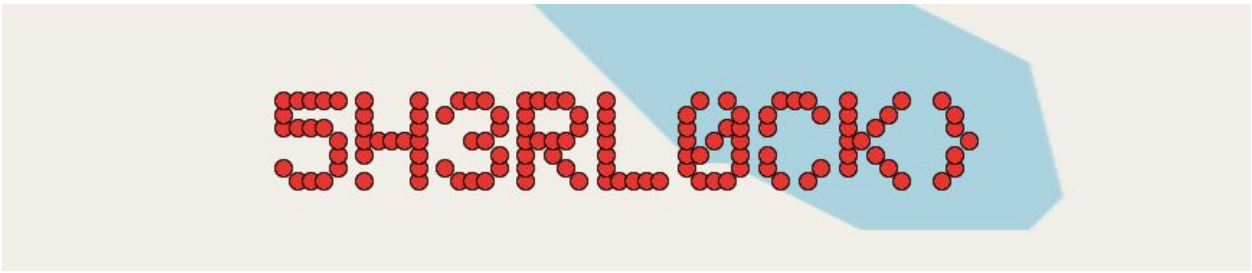
With iPhone Backup Viewer we can browse around the backup and export single files.

There was some discussion about a sports tracker called Endomondo in the WhatsApp conversation. In the applications you can see that the Endomondo Sports Tracker is installed. I extracted the endomond.db database from the application (Applications > Endomondo > Documents > Endomondo.db) and read it with DB Browser for SQLite. There were multiple tables in the database and in one of those tables (the “track points” table) I was able to find more of the coordinates.



	workoutKey	tpIndex	timestamp	instruction	lat	lng	distance	speed	altitude	horizontalAccuracy	verticalAccuracy	duration	heartRate	ca
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2	0	1578059345.7...	2	NULL	NULL	0.0	NULL	NULL	NULL	NULL	0.0	0	-1
2	2	1	1578059345.8...	-1	66.37896	28.445952	0.0	1.3366360664...	24.112560272...	8.0012083053...	16.792707443...	0.0464868545...	0	-1
3	2	2	1578059353.9...	-1	66.37926	28.445952	0.0051385597...	2.3767673969...	24.368160247...	6.0009059906...	13.099411010...	8.2068638801...	0	-1
4	2	3	1578059363.9...	-1	66.37936	28.445952	0.0154862506...	4.338294506073	23.745271682...	8.0012083053...	8.5593614578...	18.206885814...	0	-1
5	2	4	1578059374.9...	-1	66.37946	28.445952	0.0320452116...	5.5844087600...	23.938419342...	8.0012083053...	9.0018768310...	29.206786870...	0	-1
6	2	5	1578059376.9...	-1	66.37886	28.446051999...	0.0383642539...	7.4608473777...	24.110012054...	8.0012083053...	12.067753791...	31.206761360...	0	-1
7	2	6	1578059379.9...	-1	66.37926	28.446051999...	0.0440622828...	7.4608473777...	24.110012054...	8.0012083053...	12.352380752...	34.20671916008	0	-1
8	2	7	1578059384.9...	-1	66.37946	28.446051999...	0.0543966628...	6.7525854110...	24.30739402771	8.0012083053...	8.6389589309...	39.206641674...	0	-1
9	2	8	1578059390.9...	-1	66.37886	28.446151999...	0.0703633278...	9.2267951965...	24.390193939...	6.0009059906...	7.8866939544...	45.206539630...	0	-1
10	2	9	1578059395.9...	-1	66.37926	28.446151999...	0.0805516317...	7.5667500495...	24.349693298...	6.0009059906...	7.5163297653...	50.206450700...	0	-1
11	2	10	1578059400.9...	-1	66.37946	28.446151999...	0.0891258791...	5.2184967994...	24.348390579...	6.0009059906...	7.2424855232...	55.206357479...	0	-1
12	2	11	1578059406.9...	-1	66.37886	28.446251999...	0.0973529145...	5.2346148490...	24.564662933...	6.0009059906...	7.2370300292...	61.206242561...	0	-1
13	2	12	1578059410.9...	-1	66.37926	28.446251999...	0.1035393476...	5.2346148490...	24.564662933...	6.0009059906...	7.3708324432...	65.206164836...	0	-1
14	2	13	1578059419.9...	-1	66.37946	28.446251999...	0.1197029426...	6.4655475616...	24.569452285...	6.0009059906...	7.3672680854...	74.205986976...	0	-1
15	2	14	1578059422.9...	-1	66.37896	28.446351999...	0.1255111098...	6.6934413909...	24.569459915...	6.0009059906...	7.1056427955...	77.205927610...	0	-1

After copying the coordinates and tossing them into a GPS Point Plotter we get the final part of the flag.



The Final Flag

It looks like we have gathered all parts of the flag so if we toss all the gathered coordinates into the GPS Point Plotter we get the following flag:

`Flag: N1XU{Y0U_F0UND_M3_5H3RL0CK}`

NUMB3RS

Challenge description:

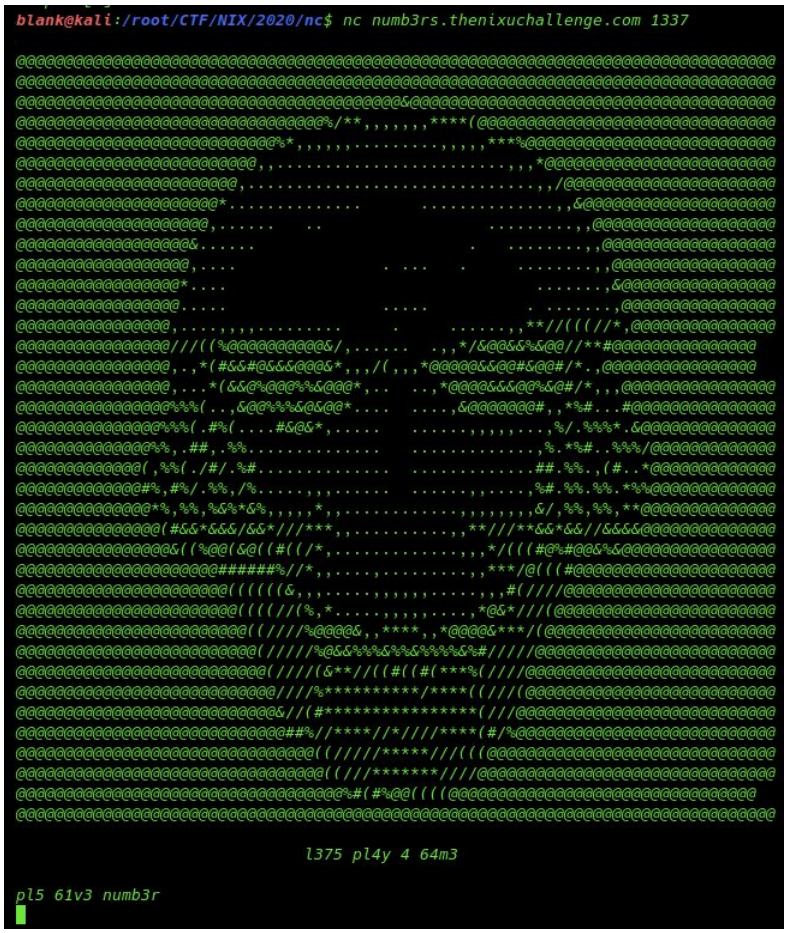
```
Hello, friend. You don't know me, but I know you. I want to
play a game. There is only one combination of numbers that
gives you the answer. Know what to do. You better hurry up.
Make your choice.

nc numb3rs.thenixuchallenge.com 1337
```

Categories: Scripting, 2020, Disobey!

The challenge gives us a host and a port to connect to using NetCat (nc).

Using nc to initiate the connection into numb3rs.thenixuchallenge.com on port 1337

We get this response:



It's asking us for a number...

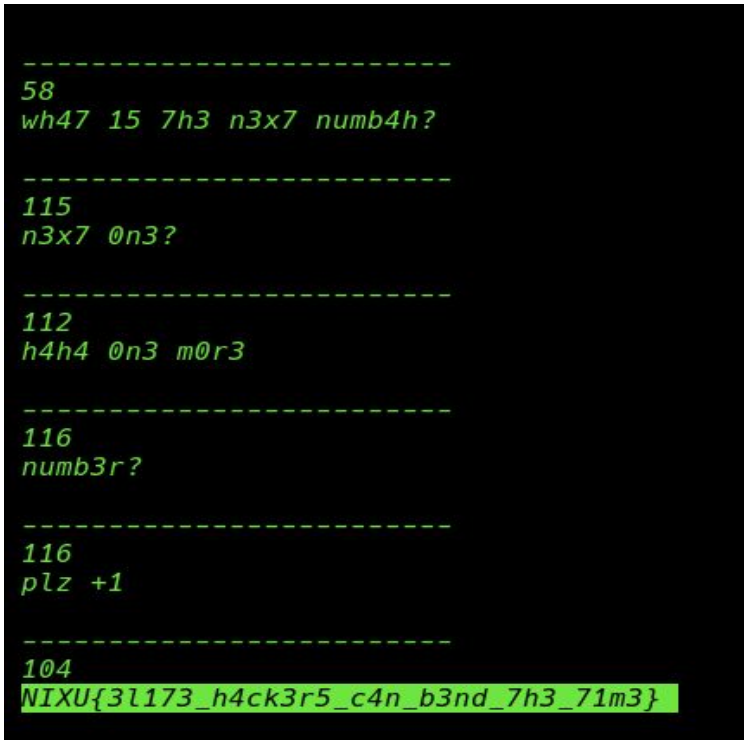
It seems like if we answer right, it asks us for the next number and there are no occurring patterns. If we answer wrong, it will give us the right number and terminate our TCP (Transmission Control Protocol) connection and we will have to start all over again.

The numbers stay the same and there are too many numbers to guess them by hand so we have to write a script to automate this task.

My coding language of choice is python due to the PWN-tools library. The PWN-tools library provides great tools for pentesting and Capture The Flag challenges.

The Script

```
#nc numb3rs.thenixuchallenge.com 1337
def reinit():
    return remote('numb3rs.thenixuchallenge.com', 1337)

sh = reinit()
#this is where our confirmed numbers will be stored
ans = ['32']
x=0
while True:
    #we need to wait 0.4 seconds before sending the numbers otherwise we
will get an "TOO FAST" Response and our connection will be terminated.
    time.sleep(0.4)
    data = sh.recv().decode()
    if '@' in data:
        print("New")
    else:
        print(data)
    try:
        sh.sendline(ans[x])
   except:
        sh.sendline('6')
#this will look at the received data. If there is a 'WR0NK!' response it'll
close the connection, grab the right number and put it into our array.
    if 'WR0NK!' in data:
        ans.append(data.split('\n')[0])
        sh.close()
        sh = reinit()
        print('+'*25)
        print(ans)
        x=0
        continue
    print('-'*25)
    x+=1


print(data)
sh.close()
```



Flag: NIXU{3l173_h4ck3r5_c4an_b3nd_7h3_71m3}