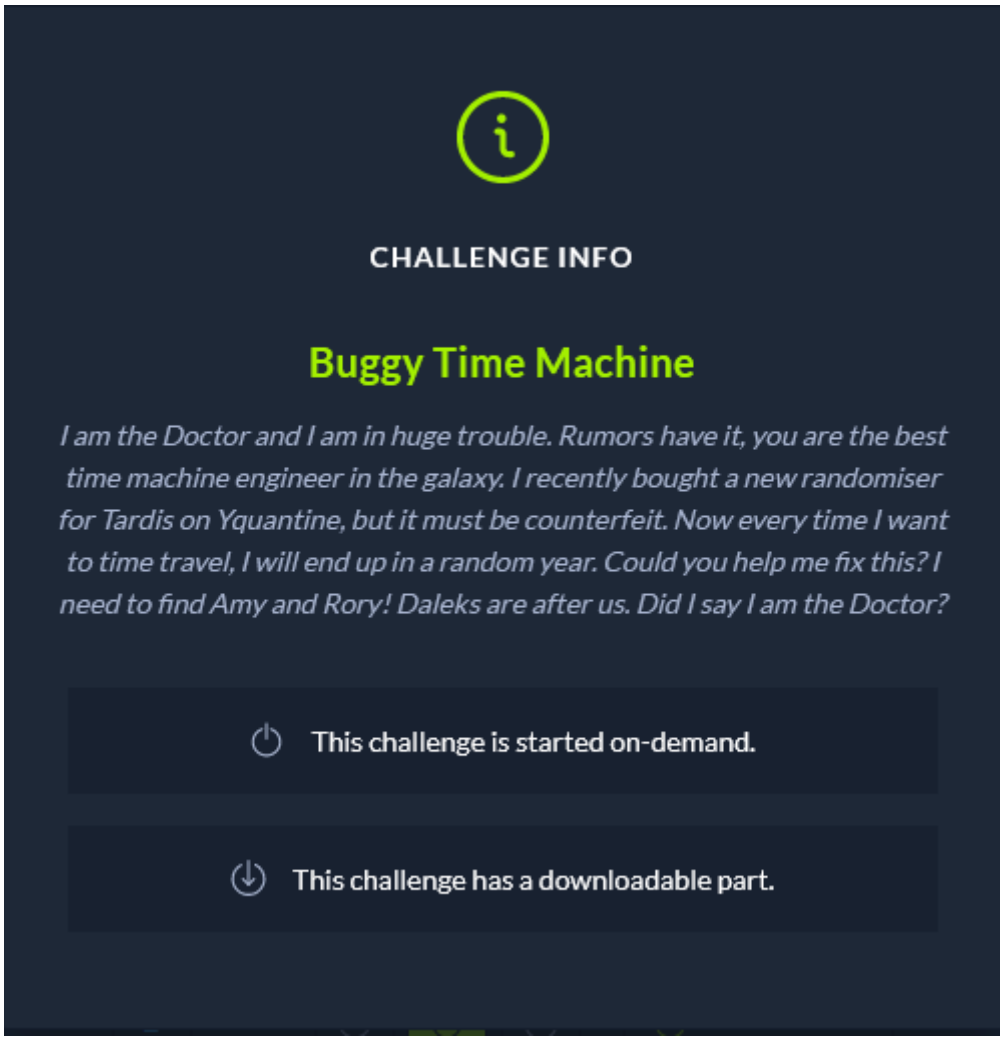
# BUGGY_TIME_MACHINE




MATERIAL:
public.py

FLAG:
HTB{l1n34r_c0n9ru3nc35_4nd_prn91Zz}

SOLVER:
M1gnus

# Foothold

The challenge presents us the code of a server:

```
import os
from datetime import datetime
from flask import Flask, render_template
from flask import request
import random
from math import gcd
import json
from secret import flag, hops, msg

class TimeMachineCore:

        n = ...
        m = ...
        c = ...


        def __init__(self, seed):
        self.year = seed

        def next(self):
        self.year = (self.year * self.m + self.c) % self.n
        return self.year

app = Flask(__name__)
a = datetime.now()
seed = int(a.strftime('%Y%m%d')) <<1337 % random.getrandbits(128)
gen = TimeMachineCore(seed)


@app.route('/next_year')
def next_year():
        return json.dumps({'year':str(gen.next())})

@app.route('/predict_year', methods = ['POST'])
def predict_year():

        prediction = request.json['year']
        try:

        if prediction ==gen.next():
                return json.dumps({'msg': msg})
        else:
                return json.dumps({'fail': 'wrong year'})

        except:

        return json.dumps({'error': 'year not found in keys.'})

@app.route('/travelTo2020', methods = ['POST'])
def travelTo2020():
        seed = request.json['seed']
        gen = TimeMachineCore(seed)
        for i in range(hops): state = gen.next()
        if state == 2020:
        return json.dumps({'flag': flag})

@app.route('/')
def home():
        return render_template('index.html')
if __name__ == '__main__':
        app.run(debug=True)
```

By analyzing the code is possible to see that the challenge deal with linear congruential generators (lcg). This kind of generators are easily breakable by only having a generated sequence of numbers. The server offers 3 API: "/next_year", "/predict_year", "/travelTo2020".
"/next_year": the response is a JSON object that contains the next number generated by the lcg.
"/predict_year": takes from a post request a year and check if the next number of the sequence is correct or not, the response contains the result of the check.
"/travelTo2020": takes from a post request a seed for the lcg, then generate n numbers, if the last one is 2020 then the flag is returned, else the last one is returned in a JSON object.

## THE ATTACK

Here is possible to find a clear explanation about how the attack to the lcg works. Once the lcg is broken n, m and c is obtained. The next step is to get the number of hops performed by travelTo2020. Is possible to use the API "/travelTo2020" sending 2020 as seed, then the final number generated from 2020 with the correct number of hops is obtained. By generating numbers from 2020 until the number returned by the API is reached is possible to obtain the number of hops used in travelTo2020. Then to get the correct seed the following operation must be performed for a number of times equal to the number of hops*:

$$x = (x * 1/m) \bmod n$$

where the first time x = 2020, then using the resulting x as seed the last hop will produce 2020 and the flag will be returned.

*the direct procedure to generate a new number is:

$$[(num * m) + c] \bmod n]$$

so if num = (num2 * 1/m) mod n then the resulting number will be num2.

# THE IMPLEMENTATION

```python
import sys
import requests
import json
import re
from gmpy2 import gcd, invert
from functools import reduce
from itertools import count


PORT = 30250
URL_FLAG = f"http://docker.hackthebox.eu:{PORT}/travelTo2020"
URL_SEQUENCE = f"http://docker.hackthebox.eu:{PORT}/next_year"
HEADERS = {"content-type": "application/json"}

def crack_unknown_increment(states, modulus, multiplier):
    increment = (states[1] - states[0]*multiplier) % modulus
    return modulus, multiplier, increment


def crack_unknown_multiplier(states, modulus):
    multiplier = (states[2] - states[1]) * invert(states[1] - states[0], modulus) % modulus
    return crack_unknown_increment(states, modulus, multiplier)


def crack_unknown_modulus(states):
    diffs = [s1 - s0 for s0, s1 in zip(states, states[1:])]
    zeroes = [t2*t0 - t1*t1 for t0, t1, t2 in zip(diffs, diffs[1:], diffs[2:])]
    modulus = abs(reduce(gcd, zeroes))
    return crack_unknown_multiplier(states, modulus)


def get_sequence():
    L = []
    for _ in range(30):
        L.append(int(requests.get(URL_SEQUENCE).json()["year"]))
    return L


def travel(x):
    payload = json.dumps({"seed": x})
    r = requests.post(URL_FLAG, data=payload, headers=HEADERS)
    return r.json()


print("Getting the sequence...")
states = get_sequence()
print("Sequence Obtained\n")

modulus, multiplier, increment = crack_unknown_modulus(states)
print(f"Modulus: {modulus}")
print(f"Multiplier: {multiplier}")
print(f"Increment: {increment}")
print()

x = 2020

print(f"Getting the hops number...")
end = travel(x)["error"]
end = int(re.findall(r" (\d+)!", end)[0])

for i in count():
    print(f"\rHops: {i}", end="")
    sys.stdout.flush()
    if x == end:
        print()
        ops = i
        break
    x = x * multiplier % modulus

x = 2020
for i in range(ops):
    x = x * int(invert(multiplier, modulus)) % modulus
    print(f"\rBreaking... {i}", end="")
x = int(x)
print()

print()
print(f"correct year: {x}")
print(f"sending the payload...")

flag = travel(x)["flag"]
print(f"flag: {flag}")
```

# Recover the flag

To recover the flag we have only to run the script:

```
C:\Users\Vittorio\Desktop\Writeups\buggy_time_machine>python lcg_attack.py
Getting the sequence...
Sequence Obtained

Modulus: 2147483647
Multiplier: 48271
Increment: 0

Getting the hops number...
Hops: 876578
Breaking... 876577

correct year: 2113508741
sending the payload...
flag: HTB{l1n34r_c0n9ru3nc35_4nd_prn91Zz}
```

# CHEESE!