Qualification Event Technical Paper

Solar Wine team

2022-06-12

SOLAR WINE

Contents

1 Revenge of the Space Math

1.1 Matters of State

- **Category**: Revenge of the Space Math
- **Points:** 35
- **Solves:** 137
- **Description:**

> I wasn't paying attention in today's Keplerian orbit mechanics class. Who wants to do my homework for me?
>
> note: It will ask you the same question a few time (randomization is busted). Just keep answering and you'll be good.

1.1.1 Write-up

Write-up by Solar Wine team

When connecting to the challenge, a set of Keplerian elements are provided. We are tasked to compute satellite's state vector from these elements:

```
$ nc matters_of_state.satellitesabove.me 5300
Ticket please:
ticket{india964294whiskey3:GK1mmXKgpFvuqWpq0tBKn48EIxj4Y4vczJDTr4sldrroibFC8PSmDv9
 ↪  das0cWbof_Q}
Orbital Elements:
Given the following orbit elements:
semi-major axis: 63780.0 km
eccentricity: 0.3
inclination: 63.0 deg
RAAN: 25.0 deg
Mean anomaly: 10.0 deg
Argument of periapsis: 78.0 deg
Time: 2022-01-01T00:00:00.000
Find the state vector of the statellite
Position: X,Y,Z
1,1,1
Velocity: Vx,Vy,Vz
1,1,1
dP: 45231.61015197461 dV: 4.868679595756561
Position incorrect
Velocity incorrect
```

The handy orbitalpy Python library has a `KeplerianElements` class which can handle this for us. The only catch is that we have to convert our values to the units that `orbitalpy` expects (meters and radians instead of kilometers and degrees), and then convert the result back before sending it to the server:

```python
import math
from astropy.time import Time
from scipy.constants import kilo
from orbital import earth, KeplerianElements

def convert(a, e, i, raan, m0, arg_pe, epoch):
    """
    :param a: Semi-major axis [km]
    :param e: Eccentricity
    :param i: Inclination [degrees]
    :param raan: Right Ascension of Ascending Node [degrees]
    :param m0: Mean anomaly [degrees]
    :param arg_pe: Argument of periapsis [degrees]
    :param epoch: Reference epoch
    """
    orbit = KeplerianElements(
        a=a * kilo,
        e=e,
        i=math.radians(i),
        raan=math.radians(raan),
        M0=math.radians(m0),
        arg_pe=math.radians(arg_pe),
        ref_epoch=Time(epoch, format="isot", scale="utc"),
        body=earth,
    )
    return [
        (orbit.r.x / kilo, orbit.r.y / kilo, orbit.r.z / kilo),
        (orbit.v.x / kilo, orbit.v.y / kilo, orbit.v.z / kilo),
    ]

pos, vel = convert(
    63780.0, 0.3, 63.0, 25.0, 10.0, 78.0, "2022-01-01T00:00:00.000"
)
print("{},{},{}".format(*pos))
print("{},{},{}".format(*vel))
```

Running the above code would yield the following result:

```
-13816.246763610543,16031.813156003369,39975.9331252851
-3.020297307426827,-1.4932804100621249,-0.15100102642583765
```

All that is left to do is to add a bit of glue to automatically send our values to the server. As was added later in the challenge's description, the server actually always sends the same orbital elements instead of generating random values each time, so we can send our result in a loop until we get the flag:

```python
from pwn import remote

def main(host, port, ticket):
    p = remote(host, port)
    p.recvline()
    p.sendline(ticket)

    while True:
        line = b""
        while line != b"Position: X,Y,Z \n":
            line = p.recvline()
            print(line)

        # Random is broken, the same challenge is always sent by the server
        pos, vel = convert(
            63780.0, 0.3, 63.0, 25.0, 10.0, 78.0, "2022-01-01T00:00:00.000"
        )
        p.sendline("{},{},{}".format(*pos))
        p.sendline("{},{},{}".format(*vel))

if __name__ == "__main__":
    main(
        "matters_of_state.satellitesabove.me",
        "5300",
        "ticket{india964294whiskey3:GK1mmXKgpFvuqWpq0tBKn48EIxj4Y4vczJDTr4sldrroib
        ↪  FC8PSmDv9das0cWbof_Q}",
    )
```

2 I Can Haz Satellite

2.1 F'DA Approved Beef

- **Category**: I Can Haz Satellite
- **Points:** 150
- **Solves:** 22
- **Description:**

2.1.1 Problem description

This challenge starts by connecting to a TCP service, which first asks for our team ticket, and then provides us with:

- an HTTP endpoint where an instance of F'Prime instance is reachable. Through this interface, we can easily send telecommands, and receive telemetry.
- a TCP endpoint for a CLI interface (which we did not understand how to use).

```
Wait for system to initialized
Challenge Web Page Starting at http://44.196.58.114:18550/
CLI available at 44.196.58.114:18551
Now F'DA Approved Beef
Get the system to return flag in telemetry to win
Time remaining to solve: 900 seconds
```

2.1.2 Recon

A command called `flagSvr.FS_FlagEnable` looks particularly interesting. It expects a filename as an argument. If we put a random filename which likely will not exist, we get the following output:

```
FlagSrvLog : Error opening flag attempt file
FlagSrv Attempt => Items Parsed: 0, Valid Items: 0, Invalid Items: 0
FlagSrv Attempt => --- Flag Not Unlocked, Fix Input and Try Again ---
```

Other commands are interesting as well:

- `fileManager.ShellCommand` expects two arguments: a shell command, and a file name where the shell command's output is to be logged. For example, running it with parameters `ls` and `out.log` is particularly interesting!

- `fileDownlink.SendFile` expects two arguments: a source filename on the satellite, and a destination filename on the simulated ground station. If run with parameters `out.log` and `out.log` after running the `fileManager.ShellCommand` above, we can get the output of the `ls` command!

At this point, we could list existing files. We saw that a file called `attempt.txt` exists, with the following contents:

```
1
2
hello
5
11
1000
167
150
21
10103
```

We did some reconnaissance on the satellite:

- We ran `find` to see what is on the filesystem.
- We ran `ps` to see what is running.
- We downloaded the script that starts everything, `/run_space.sh`. It has the following contents:

```
echo "Starting spacecraft"
cd /home/space/fsw/
echo "${FLAG}" > /home/space/fsw/.FlagData
unset FLAG
sleep 15
./satellite.exe -a 172.16.238.3 -p 50000
```

- We looked at `satellite.exe`'s environment. No flag there :-(
- `/home/space/fsw/.FlagData` does not exist at the time we issue commands: it is removed by `satellite.exe` upon startup.

It seems there are no shortcuts, we'll have to understand this C++ binary…

If `flagSvr` is run with `attempt.txt` as parameter, we get the following output:

```
FlagSrv Attempt => Loading attempt.txt file for flag enable attempt
FlagSrv Attempt => Items Parsed: 10, Valid Items: 3, Invalid Items: 6
FlagSrv Attempt => --- Flag Not Unlocked, Fix Input and Try Again ---
```

3 valid items!

By reverse engineering `satellite.exe`, we came to understand that it needs to find 2022 "valid" elements from the file that it receives as first argument. We thought that numbers satisfying the following constraints would be valid:

- `number % 3 != 0`
- `number % 5 != 0`
- `number % 7 != 0`
- `number % 6 != 5`
- `number % 6 != 1`

That sounds similar to the infamous "quasi-prime" numbers presented at Black Hat in 2019! We tried to generate a list of such numbers, but it did not yield a flag.

At this point, two options were explored in parallel:

- keep staring at that C++ binary which Ghidra does not decompile to something that is easy to understand. Reverse Engineering wizards were busy on other problems…
- try to understand which of the numbers in the provided `attempt.txt` are deemed valid.

And then… we lost track of what we really did. After a lot of trial and error, we understood that:

- a file containing "2", "5" and "11" has 3 valid items,
- a file containing "2", "5" "11" and "11" has 4 valid items. Whaaat? Can we just repeat a valid number??

2.1.3 Solution

So, we generated a file containing 2022 lines with the number 11 on them, using this script:

```
for i in seq 1 2022; do echo 11; done > out.txt
```

We uplinked it to `/home/space/fsw/a.txt`, and ran the `flagSvr.FS_FlagEnable` with `a.txt` as a parameter.

In the logs, we could see a message saying that the flag was unlocked, and that we could just download `flag.txt`. We downlinked that file to the ground, then downloaded it. And scored!

2.1.4 2nd attempt

To be able to write a proper write-up, we restarted the challenge, uploaded the file containing 2022 times the number "11", ran `flagSvr.FS_FlagEnable` … and did not get a flag! Maybe the attempts we had done before sending that file had done something to `satellite.exe`'s internal state?

We decided that this being a CTF, the only thing that matters is the flag, and turned our attention to other challenges. We look forward to reading the challenge's source code!

2.2 Fun in the Sun

- **Category**: I Can Haz Satellite
- **Points:** 76
- **Solves:** 55
- **Description:**

We are provided with the TLE of our satellite (SunFunSat) in `sat.tle` and the ephemeride DE440 in `de440s.bsp`. The remote service asks for the quaternion to point the solar panels of the SunFunSat to the sun at a precise time:

```
Ticket please:
 ticket{yankee194846zulu3:GLz-8CffBJvumO-CPzi1HE4U-hQMinZDEMLDM6uQ6CQje8mg0vcwLsrf
 ↪  h-ZPFp1DUA}
            FUN
              IN
                THE
                  SUN!!

               y
              /              .
           ////             <O>
          ////o--z           '
            |
            x


Provide the the quaternion (Qx, Qy, Qz, Qw) to point your spacecraft at the sun at
 ↪  2022-05-21 14:00:00 UTC
The solar panels face the -X axis of the spacecraft body frame or [-1,0,0]
```

The first step was to use `skyfield` to get the position of the satellite relative to the solar system's center of mass with `(planets['Earth'] + sat).at(t)` at 2022-05-21 14:00:00 UTC. Then, the `.observe(planets['Sun']).position.au` API gives the astrometric position of the sun relative to the SunFunSat:

```python
from skyfield.api import EarthSatellite, load

planets = load('de440s.bsp')

ts = load.timescale()
t = ts.utc(2022, 5, 21, 14, 00)
```

```
sat = EarthSatellite(
        '1 70003F 22700A   22140.00000000  .00000000  00000-0  00000-0 0  100',
        '2 70003  70.9464 334.7550 0003504   0.0012  16.1023 13.17057395  100',
        'Hack-A-Sat', ts)

print((planets['Earth'] + sat).at(t).observe(planets['Sun']).position.au)
```

This outputs `[0.50290776 0.80585346 0.3493443]`, that can be converted to a quaternion using the same code as in 2021:

```
import numpy as np

body = [-1, 0, 0]
b = [0.50290776, 0.80585346, 0.3493443]

axis = np.cross(body, b)
axis /= np.linalg.norm(axis)
arc = np.dot(body, b)

qr = np.sqrt((1 + arc) / 2)
qi = np.sqrt((1 - arc) / 2) * axis

print(f'Qx = {qi[0]} Qy = {qi[1]} Qz = {qi[2]} Qw = {qr}')
```

The server accepted these values and gave us the flag:

```
Ticket please:
 ticket{yankee194846zulu3:GLz-8CffBJvumO-CPzi1HE4U-hQMinZDEMLDM6uQ6CQje8mg0vcwLsrf
 ↪   h-ZPFp1DUA}
            FUN
              IN
                THE
                  SUN!!

               y
              /              .
           ////             <O>
          ////o--z           '
            |
            x


Provide the the quaternion (Qx, Qy, Qz, Qw) to point your spacecraft at the sun at
 ↪  2022-05-21 14:00:00 UTC
```

```
The solar panels face the -X axis of the spacecraft body frame or [-1,0,0]
Qx = 0.0
Qy = 0.34478899978532596
Qz = -0.7953454756437823
Qw = 0.4985440000641869
Quaternion normalized to: [ 0.          0.344789   -0.79534548  0.498544  ]
The solar panels are facing 0.395 degrees away from the sun
You got it! Here's your flag:
flag{yankee194846zulu3:GN6v4GSL9Hl9lORAFu8_tedD-aWYrJplfXgGSJcRL9l5muYGX8myZVqWpJf
↪  AdBEyawt9rQGCb2FEtTj_4R7kFFA}
```

2.3 Red Alert

- **Category**: I Can Haz Satellite
- **Points:** 54
- **Solves:** 82
- **Description:**

2.3.1 Problem description

This challenge starts by connecting to a TCP service at `red_alert.satellitesabove.me:5100`, which first asks for our team ticket, and then provides us with an HTTP endpoint where a Grafana instance is reachable. Credentials for the instance is provided in the problem description

The problem description indicates that the satellite is controlled through an API server, which the Grafana server can reach at `groundstation:5000`. It has 5 commands:

- one called `play` to start the mission,
- two commands called `laser-on` and `laser-off` to enable/disable obstacle detection,
- two commands called `detector-on` and `detector-off` to enable/disable lasers, which will pulverize obstacles and clear the path

2.3.2 Recon

At first, we noticed that the Grafana instance was provisionned with:

- one PostgreSQL data source,
- only one custom dashboard showing a single graph where the battery level is displayed.

That reminded us of the battery management problem during Hack-a-Sat2 finals :)

Situational awareness

We modified the dashboard to add more data, which autocompletion on Grafana helped us discover:

- from the `lasers` table: `count`, `cycle`, `state`
- from the `cdh` table: `count`, `battery`, `temp`. `cdh` probably stands for Command & Data Handling; the most interesting metric was `battery`

- from the `detector` table: `count`, `cycle`, `range`, `state`. The most interesting metric was `range`.

The dashboard we created is available here if you are reading this from the GitHub repository.

Interacting with the API server

From the problem description, we knew that we had to issue queries to `groundstation:5000`. We had to make grafana somehow issue those requests. We first tried adding new data sources, looking for a type that would allow querying an HTTP server, even though the Data Source would not be valid. This did not work with any of the available data types!

Then, looking more carefully at the grafana server's configuration, we found out that the "Alerting" section was conveniently provisioned to send alerts to "contact points", 5 of which were configured, one for each API endpoint. Cherry on top: a "test" button makes Grafana issue queries. We're ready to start!

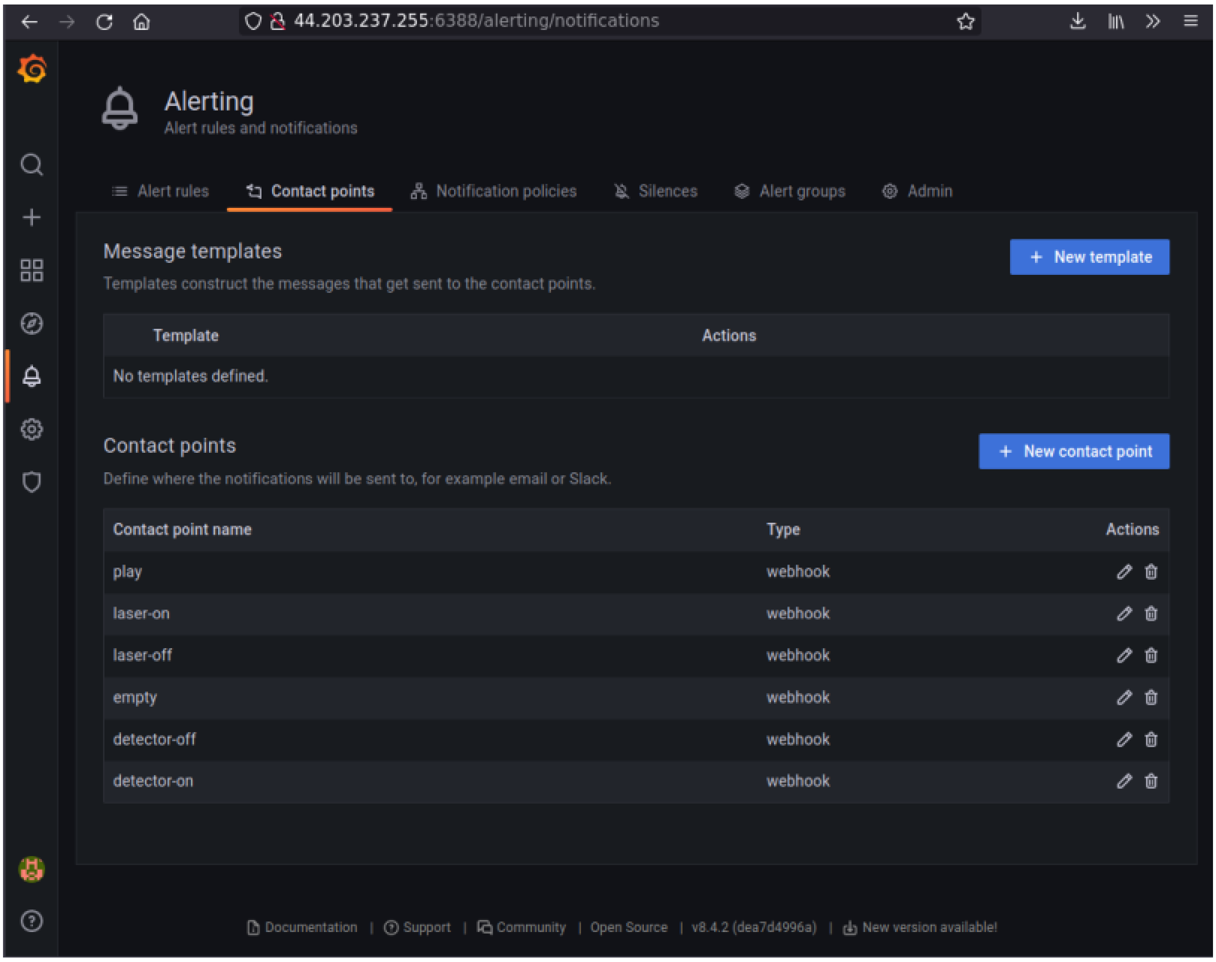

Figure 1: Contact points

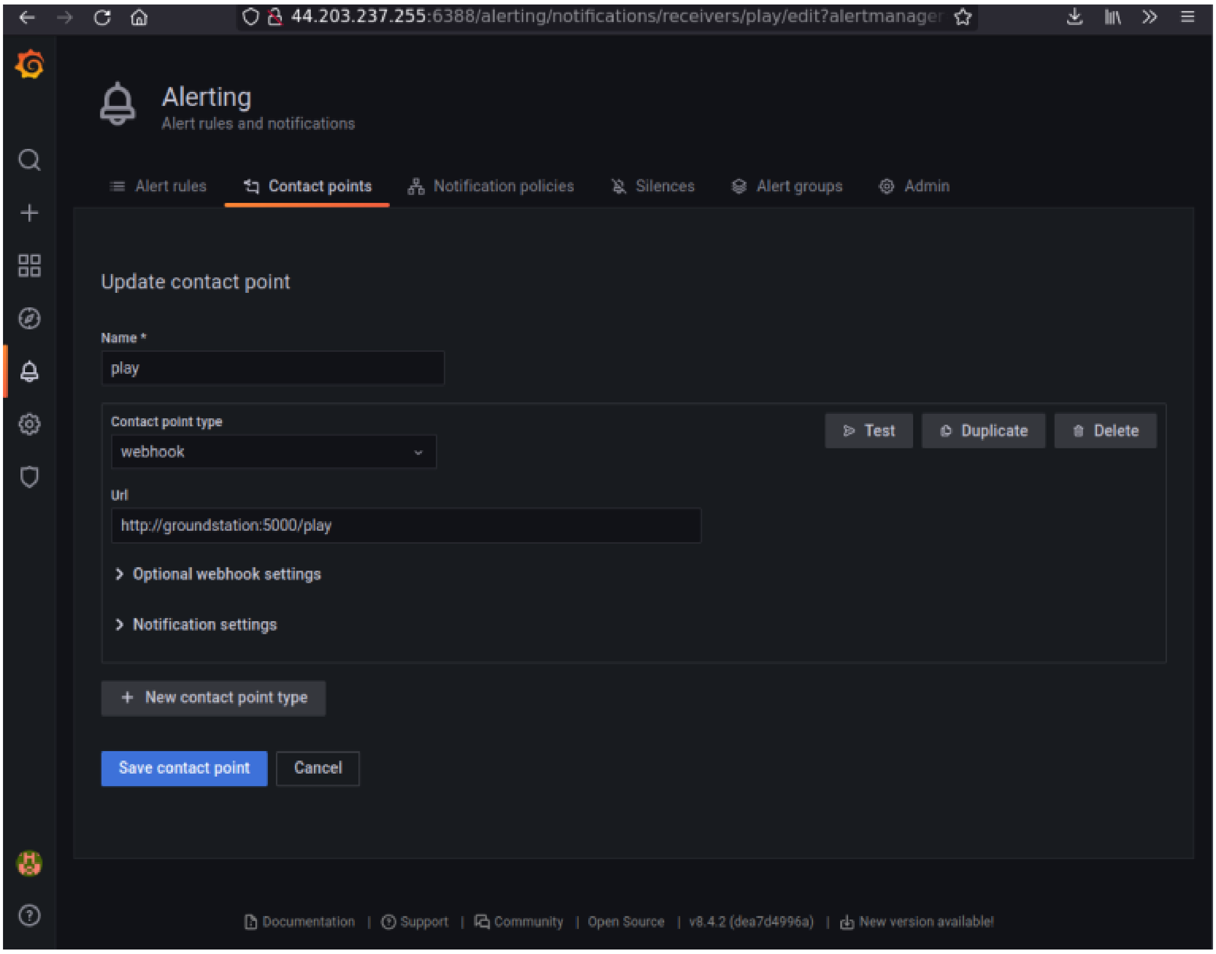

Figure 2: Interface to trigger WebHook queries

We then queried the `play` endpoint… After some time, this message appeared on the console where `nc red_alert.satellitesabove.me 5100` was running:

```
groundstation  | Satellite destroyed
groundstation  | MISSION FAILED
```

2.3.3 Game plan

After experimenting, we understood that:

- the detector and lasers both use battery,
- the battery recharges when lasers are not used, and recharges faster when the detector is not used either,

- the detector reports a distance to the next obstacle, starting when the obstacle is less than 5000 units away,
- lasers can destroy obstacles when it is closer than 1000 units,
- it takes about 21 simulated minutes for obstacles to move from a distance of 5000 units to a distance of 1000 units.

Our strategy was then as follows:

- start the game,
- periodically enable the detector, at least once every 20 simulated minutes,
- if it reports a distance of 5000: there is no obstacle, wait,
- if it reports a distance of less than 1000: wait for the distance to drop down to about 2000, then enable lasers until a message indicating that an obstacle was destroyed appears on the console.

This strategy allows us to save enough energy for batteries to recharge while we destroy all obstacles.

2.3.4 Execution

We noticed that this strategy can be automated by creating a few clever alerting rules in grafana. Of course, manually executing it was much quicker than writing automated rules. Here is our dashboard after a winning attempt:

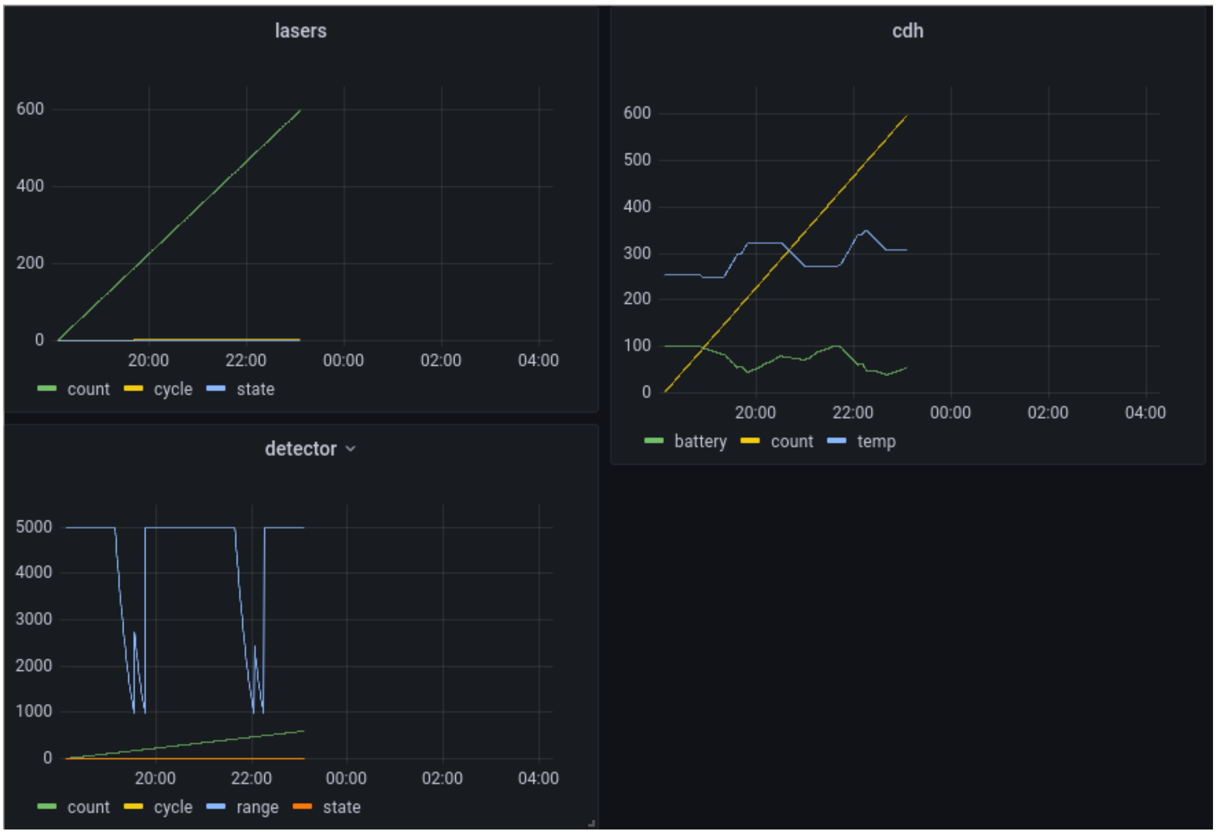

Figure 3: Dashboard showing a successful attempt

After about 5 simulated hours, a flag appeared on the console:

```
groundstation  | You survived!
groundstation  | flag{delta692703victor3:GBYwGCd_rt-oltlvIe3WfWP_RKR_mB5NBahrGIFTc⌋
 ↪  yKSLvXtyMXSidYxGEKZzs6huo_jXrzAFv4zKW9328ig67s}
```

3 We Get Signal

3.1 Power Point

- **Category**: We Get Signal
- **Points**: 165
- **Solves**: 19
- **Description:**

> Break out Microsoft Office because its time to make some slides… just kidding!

> There is a satellite out there transmitting a flag. You are on a moving platform and you don't know how you will be moving or where the transmitter is or how the transmitter is moving. Luckily you have a steerable antenna! To get the flag:
>
> 1. Find the transmitter by steering the antenna.
> 2. Keep the antenna pointed at the transmitter and try to maximize signal power.
> 3. Decode the samples coming in from your receiver.
>
> Your co-worker told you they think the satellites is "North-ish and sort of close to the horizon"

> Send az, el commands to the antenna; an antenna command is formatted as a string for example if az=77.1 and el=73.2
>
> `send: 77.10,22.2\n`
>
> Please send one command at a time.
>
> After you send a command the receiver will reply with a fixed number of IQ samples via TCP.

3.1.1 Requirements

This write-up will use:

- Python3
- Numpy: https://numpy.org/install/

3.1.2 Write-up

Write-up by Solar Wine team

First, connect to the challenge via nc. This is what the service asks us when we provide it with our team ticket:

```
$ echo 'ticket{juliet365159quebec3:GEwdB3WJC6Lzj_rW7wn3Sine6taWUC79-XDO_MRl8oyg3MN
↪ SlY-_c5pgmM6bqKTlSw}' | nc power_point.satellitesabove.me
↪ 5100
Ticket please:
Keep the signal power high if to get the flag



Antenna pointing TCP server accepts commands at 44.193.213.139:12688
Sample TCP server will provide samples at 44.193.213.139:12689
The flag begins at byte: 7200
Sample server: Waiting for client connection
```

After finding a good start position for the antenna we tried our best to keep it in the right direction. Unfortunately we had some bad signal that we fixed manually. To keep the antenna pointing in the right direction we analyzed the strength of the signal:

```python
## calculate strength of signal
def getSigStrength(buf):
    dt = np.dtype(np.float32)
    dt = dt.newbyteorder('<')

    data = np.frombuffer(buf, dtype=dt)
    IQ = data[0::2] +1j* data[1::2]
    IQ = IQ[:1024*8]

    s = 100e2
    N = 1024
    PSD = (np.abs(np.fft.fft(IQ))/N)**2
    PSD_log = 10.0*np.log10(PSD)
    PSD_shifted = np.fft.fftshift(PSD_log)
    IQ = IQ * np.hamming(len(IQ))

    Fs = 100e3
    center_freq = 0                      # frequency we tuned our SDR to
    f = np.arange(Fs/-2.0, Fs/2.0, Fs/N) # start, stop, step.  centered around 0 Hz
    f += center_freq                     # now add center frequency

    sig_freq=f[np.argmax(PSD_shifted)]
    mx = np.max(PSD_shifted)

    return (mx,sig_freq)
```

Using the signal strength, we first scan around the estimated position (around the north and close to horizon). This provides a good starting point, at az=8.0°,el=19.0°; with a pretty large margin.

By analysing the signal using GNU Radio (waterfall, amplitude over time, phase over time and constellation diagram), we can identify the modulation and the “baudrate” : QPSK is used with a 2-bit symbol sent every 10 points. Nevertheless, the symbol encoding is still not know yet.

From the console, we know that the flag is sent at the offset 7200. Nevertheless, since the satellite is moving, we need to implement a tracking mechanism to get more data.

The implemented tracking algorithm move 2.0° on both direction to update the receiving point using the maximum strength value. There is some data loss, but prior the flag.

```
bestPoint=(80,190)
while True:
    bestSig = 0
    for azo in range(-1,2,1):
        (az,el) = (bestPoint[0]+(azo*20), bestPoint[1])
        buf = moveAntenna(az, el)

        if(buf == None):
            print("%04x %.1f,%.1f bad signal" %(N, az,el))
            continue

        (snr,strength,sig_freq) = getSigStrength(buf)
        if(strength > bestSig):
            bestSig = strength
            bestPoint = (az,el)

        dec = decode(buf)

        print("%04x %.1f,%.1f %s (%.1f)" % (N, az,el, dec, strength))

    for elo in range(-1,2,1):
        (az,el) = (bestPoint[0], bestPoint[1]+(elo*20))
        buf = moveAntenna(az, el)

        if(buf == None):
            print("%04x %.1f,%.1f bad signal" %(N, az,el))
            continue

        (snr,strength,sig_freq) = getSigStrength(buf)
        if(strength > bestSig):
            bestSig = strength
```

```
            bestPoint = (az,el)

        dec = decode(buf)

        print("%04x %.1f,%.1f %s (%.1f)" % (N, az,el, dec, strength))
```

Next, we retrieved data, decoded it, and reversed the encoding to finally get the flag.

Running the python script, we get:

```
$ python solver.py
b'Sending ... > ticket{juliet365159quebec3:GEwdB3WJC6Lzj_rW7wn3Sine6taWUC79-XDO_MR
↪  l8oyg3MNSlY-_c5pgmM6bqKTlSw}'
b'44.193.213.139' b'20611'
b'44.193.213.139' b'20610'
7200
b'Sample server: Waiting for client connection\nSample Server: Client
↪  connected\nAntenna pointing server: Waiting for client to connect\nAntenna
↪  pointing server: Client connected\nPress Enter to quit:'
0000 > [60.0, 190.0] (29.9)
001c > [60.0, 190.0] (30.3)
0038 > [80.0, 190.0] (37.9)
...
HackASat\nHackASat\nHackASat\nHackASat\nHackASat\nHackASat\nHackASat\nHackASat\nHa
↪  ckASat\nHackASat\nHackASat\nHackASat\nHackASat\nHackASat\nflag{juliet365159que
↪  bec3:GHsLDtL_uXX91-6xVhUQGzZq9G4SbtJrjvYIc3Y_r-ydrIv7B8myunQwW5-B1l-tzACqnBOgW
↪  yZ8UlMzWzp1QMc}\n
```

4 Crypto Category Placeholder Name

4.1 Leggo my Steggo!

- **Category**: Crypto Category Placeholder Name
- **Points:** 121
- **Solves:** 30
- **Description:**

> We've been running this satellite that has the same command mnemonic as an old Hack-a-Sat 2020 satellite - but now its infected with something.
>
> In our last communication it dumped out all these files.
>
> Re-gain authority over the satellite
>
> The satellite seems to still respond to the commands mnemonics we used before but replies in a strange way.
>
> https://github.com/cromulencellc/hackasat-final-2020/tree/master/solvers/c4/solver_ruby_scripts

> You'll need these files to solve the challenge.
>
> - https://static.2022.hackasat.com/e57q8n0po2v67dglp44q7j6kqlov

4.1.1 Write-up

Write-up by Solar Wine team

First, connect to the challenge via nc. This is what the service asks us when we provide it with our team ticket:

```
$ echo 'ticket{uniform258749delta3:GM36VW2ZPK-zL-h_r1CMzHsQyBecBS2NGFK5CialLzOZNp1
↪  TyOnRc_1qzXV3k_meLA}' | nc leggo.satellitesabove.me
↪  5300
Ticket please:
Can you regain control!
CMD>some_garbage_command
----INVALID------
```

This challenge ask us to re-gain authority over the satellite. We tried several commands but nothing worked.

Inside the provided archive we found 10 satellite pictures The title of the challenge contains `steggo`, so maybe there is some steganography involved.

```
$ steghide extract -sf boston.jpg -xf out.txt
Entrez la passphrase:
Ecriture des donnees extraites dans "out.txt".
$ cat out.txt
4aada52c9abb92d32c57007091f35bc8
```

There is no passphrase, just let it empty. Repeating the operation for each file and we find:

```
boston   > 4aada52c9abb92d32c57007091f35bc8
chicago  > a6ec1dcd107fc4b5373003db4ed901f3
la       > ae7a2a6b7b583aa9cee67bd13edb211e
miami    > 1312798c840210cd7d18cf7d1ff64a40
nyc      > e115d6633cdecdad8c5c84ee8d784a55
oahu     > Use the command mnemonic that outputs moon.png
portland > 20b7542ea7e35bf58e9c2091e370a77d
sf       > f24b6a952d3f0a8f724e3c70de2e250c
slc      > 09e829c63aff7b72cb849a92bf7e7b48
vegas    > 7cef397dfa73de3dfedb7e537ed8bf03
```

Output seems to be a MD5 hash. We could reverse hashes with https://crackstation.net/

```
Hash                              Type  Result
4aada52c9abb92d32c57007091f35bc8  md5   beanpot
a6ec1dcd107fc4b5373003db4ed901f3  md5   deepdish
ae7a2a6b7b583aa9cee67bd13edb211e  md5   cityofangels
1312798c840210cd7d18cf7d1ff64a40  md5   springbreak
e115d6633cdecdad8c5c84ee8d784a55  md5   fuhgeddaboudit
20b7542ea7e35bf58e9c2091e370a77d  md5   stayweird
f24b6a952d3f0a8f724e3c70de2e250c  md5   housingmarket
09e829c63aff7b72cb849a92bf7e7b48  md5   skiparadise
7cef397dfa73de3dfedb7e537ed8bf03  md5   letsgolasvegas
```

Following the link provided in the description, https://github.com/cromulencellc/hackasat-final-2020/tree/master/solvers/c4/solver_ruby_scripts and the hint `Use the command mnemonic that outputs moon.png`, we tried several commands on the satellite. Only `TAKE_IMG` and `PLAYBACK_FILE` seemed to work, the satellite answered with a question, a string that looks like a SHA256 hash.

```
CMD>PLAYBACK_FILE
04ca7d835e92ae1e4b6abc44fa2f78f6490e0058427fcb0580dbdcf7b15bbb55?
>>0
Wrong
```

```
CMD>TAKE_IMG
983b1cc802ff33ab1ceae992591f55244538a509cd58f59ceee4f73b6a17b182?
>>
```

Investigate the pictures inside the archive again and you will find that the asked sha256 match a picture:

```
$ sha256sum *
d6bc6fbee628c3278ef534fd22700ea4017914c2214aa86447805f858d9b8ad4  boston.jpg
04ca7d835e92ae1e4b6abc44fa2f78f6490e0058427fcb0580dbdcf7b15bbb55  chicago.jpg
242f693263d0bcc3dd3d710409c22673e5b6a58c1a1d16ed2e278a8d844d7b0b  la.jpg
2aa0736e657a05244e0f8a1c10c4492dde39907c032dba9f3527b49873f1d534  miami.jpg
983b1cc802ff33ab1ceae992591f55244538a509cd58f59ceee4f73b6a17b182  nyc.jpg
7e03349fe5fa4f9e56642e6787a0bfda27fb4f647e3c283faaf7bd91dbfd1d39  oahu.jpg
b4447c4b264b52674e9e1c8113e5f29b5adf3ee4024ccae134c2d12e1b158737  portland.jpg
f37e36824f6154287818e6fde8a4e3ca56c6fea26133aba28198fe4a5b67e1a1  sf.jpg
088f26f7c0df055b6d1ce736f6d5ffc98242b752bcc72f98a1a20ef3645d05c1  slc.jpg
3b20a3b5b327c524674ca5a8310beb2d9efc5c257e60c4a9b0709d41e63584a3  vegas.jpg
```

Put all together and provide the reversed MD5 hash of the picture as an answer to the satellite's question. Unfortunately, we didn't get the flag with the `TAKE_IMG` command:

```
$ echo 'ticket{uniform258749delta3:GM36VW2ZPK-zL-h_r1CMzHsQyBecBS2NGFK5CialLzOZNp1
 ↪  TyOnRc_1qzXV3k_meLA}' | nc leggo.satellitesabove.me
 ↪  5300
Ticket please:
Can you regain control!
CMD>TAKE_IMG
983b1cc802ff33ab1ceae992591f55244538a509cd58f59ceee4f73b6a17b182?
>>fuhgeddaboudit
Thank you
f37e36824f6154287818e6fde8a4e3ca56c6fea26133aba28198fe4a5b67e1a1?
>>housingmarket
Thank you
3b20a3b5b327c524674ca5a8310beb2d9efc5c257e60c4a9b0709d41e63584a3?
>>letsgolasvegas
Thank you
```

```
088f26f7c0df055b6d1ce736f6d5ffc98242b752bcc72f98a1a20ef3645d05c1?
>>skiparadise
Thank you
d6bc6fbee628c3278ef534fd22700ea4017914c2214aa86447805f858d9b8ad4?
>>beanpot
Thank you
You did it
I'm still not giving you the satellite back
```

But we got the flag with the `PLAYBACK_FILE` command:

```
$ echo 'ticket{uniform258749delta3:GM36VW2ZPK-zL-h_r1CMzHsQyBecBS2NGFK5CialLzOZNp1
↪ TyOnRc_1qzXV3k_meLA}' | nc leggo.satellitesabove.me
↪ 5300
Ticket please:
Can you regain control!
CMD>PLAYBACK_FILE
3b20a3b5b327c524674ca5a8310beb2d9efc5c257e60c4a9b0709d41e63584a3?
>>letsgolasvegas
Thank you
242f693263d0bcc3dd3d710409c22673e5b6a58c1a1d16ed2e278a8d844d7b0b?
>>cityofangels
Thank you
04ca7d835e92ae1e4b6abc44fa2f78f6490e0058427fcb0580dbdcf7b15bbb55?
>>deepdish
Thank you
b4447c4b264b52674e9e1c8113e5f29b5adf3ee4024ccae134c2d12e1b158737?
>>stayweird
Thank you
f37e36824f6154287818e6fde8a4e3ca56c6fea26133aba28198fe4a5b67e1a1?
>>housingmarket
Thank you
OK you can have the satellite back
flag{uniform258749delta3:GJs9lhhGq_0kjQfvR6-78J851hO5zmFdtDDM5Nb83dAMW9QvVhlxM13Fv
↪ E1FSYSAvnnPCxoXicV4Sa2hpw5JZfg}
```

5 The Only Good Bug is a Dead Bug

5.1 Bit Flipper

- **Category**: The Only Good Bug is a Dead Bug
- **Points:** 257
- **Solves:** 8
- **Description:**

> We're trying to send our satellite a reset, but some bad guy flashed our reset module with a program to make it really difficult.
>
> We were only able to recover the binary off their computer… Please help.

> You'll need these files to solve the challenge.
>
> https://static.2022.hackasat.com/986wf5z8nf1jax51bmfmhq4n5rdy

5.1.1 Write-up

Write-up by Solar Wine team

The program is using a 32-bit SPARC instruction set to implement a game where the player has a guess a random 16-bit number:

```
Level 1. FIGHT

Guess: 1
15
Guess: 5
15
Guess: 20
15
Guess: 20
15
```

The number which is returned is a difficulty, which is read from a file named `hardest_rotations.nums` which is not provided. In fact the guess is not really "just a guess": it is a number which is XOR-ed with the number to be guessed (with a constant 16-bit rotation).

After some tries, a first invariant became clear: if a number has an odd number of 1 in its binary representation, its difficulty is 16. Then, trying to flip each of the 16 bits separately led to the following experimentation result:

```
input -> xor-cumulated input -> difficulty returned by the remote server
0x0001 -> query 0x0001 -> 15
0x0003 -> query 0x0002 -> 12
0x0006 -> query 0x0004 -> 15
0x000c -> query 0x0008 -> 14
0x0018 -> query 0x0010 -> 15
0x0030 -> query 0x0020 ->  9
0x0060 -> query 0x0040 -> 15
0x00c0 -> query 0x0080 -> 14
0x0180 -> query 0x0100 -> 15
0x0300 -> query 0x0200 -> 12
0x0600 -> query 0x0400 -> 15
0x0c00 -> query 0x0800 -> 14
0x1800 -> query 0x1000 -> 15
0x3000 -> query 0x2000 ->  9
0x6000 -> query 0x4000 -> 15
0xc000 -> query 0x8000 -> 14
```

In this table, “15” alternates every two positions and “14” every four positions. This gives a strong indication that the content of `hardest_rotations.nums` is very structured.

After some attempts, we successfully managed to generate a file which shares this behavior:

```python
import itertools

def rol16(value: int, shift: int) -> int:
    return ((value << shift) & 0xffff) | (value >> (16 - shift))

hardest_rotations = [None] * 0x10000
for iter_count in range(16):
    base_value = hardest_rotations.index(None, 1)
    possible_shifts = set()
    for bit1 in range(16):
        for bit2 in range(bit1 + (1 << iter_count), 16, 1 << iter_count):
            possible_shifts.add((1 << bit1) ^ (1 << bit2))
        possible_shifts.add(rol16(base_value, bit1) ^ base_value)
    for val1, val2 in itertools.combinations(possible_shifts, 2):
        new_shift = val1 ^ val2
        if new_shift not in possible_shifts:
            possible_shifts.add(new_shift)
            for val in list(possible_shifts):
```

```
                possible_shifts.add(new_shift ^ val)

    for shift in possible_shifts:
        assert hardest_rotations[base_value ^ shift] is None
        hardest_rotations[base_value ^ shift] = 16 - iter_count

    print(f"{iter_count} (difficulty {16 - iter_count}): from {base_value:#x}:
     ↪  {len(possible_shifts)}={len(possible_shifts):#x} values")

with open("hardest_rotations.nums", "wb") as f:
    for diff in hardest_rotations[1:]:
        f.write(f"{diff:02d}\n".encode())
```

While generating the file, this script displayed the number which was used to initiate the search of each difficulty level:

```
 0 (difficulty 16): from 0x1: 32768=0x8000 values
 1 (difficulty 15): from 0x3: 16384=0x4000 values
 2 (difficulty 14): from 0x5: 8192=0x2000 values
 3 (difficulty 13): from 0xf: 4096=0x1000 values
 4 (difficulty 12): from 0x11: 2048=0x800 values
 5 (difficulty 11): from 0x33: 1024=0x400 values
 6 (difficulty 10): from 0x55: 512=0x200 values
 7 (difficulty  9): from 0xff: 256=0x100 values
 8 (difficulty  8): from 0x101: 128=0x80 values
 9 (difficulty  7): from 0x303: 64=0x40 values
10 (difficulty  6): from 0x505: 32=0x20 values
11 (difficulty  5): from 0xf0f: 16=0x10 values
12 (difficulty  4): from 0x1111: 8=0x8 values
13 (difficulty  3): from 0x3333: 4=0x4 values
14 (difficulty  2): from 0x5555: 2=0x2 values
15 (difficulty  1): from 0xffff: 1=0x1 values
```

This gave a winning strategy for the game: according to the difficulty returned by the server, enter the given number!

```
guess_for_value = {
    16: 1,       15: 3,       14: 5,       13: 0xf,
    12: 0x11,    11: 0x33,    10: 0x55,     9: 0xff,
     8: 0x101,    7: 0x303,    6: 0x505,    5: 0xf0f,
     4: 0x1111,   3: 0x3333,   2: 0x5555,   1: 0xffff,
}
```

With this, getting the flag only required playing the game three times.

```
Level 1. FIGHT
    0x0001 -> 13
    0x000f -> 11
    0x0033 ->  6
    0x0505 ->  5
    0x0f0f ->  4
    0x1111 ->  2
    0x5555 ->  1
    0xffff -> FOUND
Reset sequence round 1 completed.
Level 2. FIGHT
    0x0001 -> 13
    0x000f -> 12
    0x0011 -> 11
    0x0033 -> 10
    0x0055 ->  8
    0x0101 ->  7
    0x0303 ->  6
    0x0505 ->  5
    0x0f0f -> FOUND
Reset sequence round 2 completed.
Level 3. FIGHT
    0x0001 -> 15
    0x0003 -> 14
    0x0005 -> 13
    0x000f -> 12
    0x0011 -> 11
    0x0033 -> 10
    0x0055 ->  9
    0x00ff ->  8
    0x0101 ->  7
    0x0303 ->  6
    0x0505 ->  5
    0x0f0f ->  4
    0x1111 ->  3
    0x3333 ->  2
    0x5555 ->  1
    0xffff -> FOUND
Reset sequence round 3 completed.
Resetting satellite: flag{uniform645260quebec3:GNkVBa64TqKVsJhK6Ss99Cw2gPnH2Gx-eWE
 ↪  jZ87o6BHrwH74eRwQDKT67Gkun3p84U-3BGaW9JstLgXeaNjwlpE}
```

5.2 It's A Wrap

- **Category**: The Only Good Bug is a Dead Bug
- **Points:** 43
- **Solves:** 108
- **Description:**

> Do you like inception?

> You'll need these files to solve the challenge.
>
> - https://static.2022.hackasat.com/n50xglah3x4cbisdicrwnc2iw4sq

5.2.1 Write-up

Write-up by Solar Wine team

First, connect to the challenge using nc. This is what the service asks us when we provide it with our team ticket:

```
$ echo 'ticket{victor657699zulu3:GNTSOYfh-bXQ7Wh4tuW0mJ8tqTaHzmOWioZAsxu5A9LC1CIpv
↪  apSHV-olGPqCCLj8Q}' | nc its_a_wrap.satellitesabove.me
↪  5300
Ticket please:
You have inherited a simple encryption dll from previous project
that you must use.  It used to work but now it seems it does not.
Given the encryption matrix:
| -3 -3 -4 |
|  0  1  1 |
|  4  3  4 |
...and the phrase 'HACKASAT3'.

 Enter the phrase in standard decimal ASCII
 and the resulting encrypted list
    Example: 'XYZ': 88,89,90
             Encrypted list: 1,2,3,4,5,6,7,8,9
 Entered phrase:  1,2,3,4,5,6,7,8,9
 Encrypted phrase:  1,2,3,4,5,6,7,8,9
 Entered phrase:  1,2,3,4,5,6,7,8,9
 Encrypted phrase:  1,2,3,4,5,6,7,8,9

Computing encrypted phrase...
```

Unfortunately, it seems that the challenge was broken. It accepts any phrase to encrypt and not only the "HACKASAT3" asked one. While some of us were decompiling the .pyc file and trying to implement the matrix cryptosystem, one of us were able to validate the challenge by giving it a 0's decimal ASCII phrase.

```
Enter phrase > 0,0,0,0,0,0,0,0,0
Enter encrypted phrase> 0,0,0,0,0,0,0,0,0
```

This phrase and encrypted phrase validated the challenge.

```
$ echo 'ticket{victor657699zulu3:GNTSOYfh-bXQ7Wh4tuW0mJ8tqTaHzmOWioZAsxu5A9LC1CIpv
↪  apSHV-olGPqCCLj8Q}' | nc its_a_wrap.satellitesabove.me
↪  5300
Ticket please:
You have inherited a simple encryption dll from previous project
that you must use.  It used to work but now it seems it does not.
Given the encryption matrix:
| -3 -3 -4 |
|  0  1  1 |
|  4  3  4 |
...and the phrase 'HACKASAT3'.

 Enter the phrase in standard decimal ASCII
 and the resulting encrypted list
    Example: 'XYZ': 88,89,90
             Encrypted list: 1,2,3,4,5,6,7,8,9
 Enter phrase> 0,0,0,0,0,0,0,0,0
 Enter encrypted phrase> 0,0,0,0,0,0,0,0,0
 Entered phrase:  0,0,0,0,0,0,0,0,0
 Encrypted phrase:  0,0,0,0,0,0,0,0,0

Computing encrypted phrase...

----------------------------------
Congradulations you got it right!!

Here is your flag:  flag{victor657699zulu3:GIW6DSLTLAeZDiA9uizlnjdlxUJkuDRs1SEoNBb
↪  3od2Yn9zdX7QYORlWwAQaq8iW5v2LO4bGWwv3thXSGR1s0jo}
----------------------------------
```

5.3 Small Hashes Anyways

- **Category:** The Only Good Bug is a Dead Bug
- **Points:** 73
- **Solves:** 58
- **Description:**

> Micro hashes for micro blaze ¯_(ツ)_/¯

> You'll need these files to solve the challenge.
>
> - https://generated.2022.hackasat.com/small_hashes_anyways/small_hashes_anyways-yankee957187romeo3.tar.bz2
> - https://static.2022.hackasat.com/hcx6yv4mw0sgrax3kea36v5vy1vi

5.3.1 Requirements

This writeup will use:

- Python3
- pwntools: https://github.com/Gallopsled/pwntools

5.3.2 Write-up

Write-up by Solar Wine team

The challenge is an ELF MicroBlaze 32-bit. We were also provided an archive with all the libraries required to run this program.

```
$ file small_hashes_anyways
small_hashes_anyways: ELF 32-bit MSB executable, Xilinx MicroBlaze 32-bit RISC,
 ↪  version 1 (SYSV), dynamically linked, interpreter /lib/ld.so.1, for GNU/Linux
 ↪  3.2.0, stripped
```

To run the program, use qemu and link to it all the libraries.

```
$ QEMU_LD_PREFIX=./microblaze-linux LD_LIBRARY_PATH=./microblaze-linux/lib
 ↪  qemu-microblaze ./small_hashes_anyways
small hashes anyways:
test
wrong length wanted 112 got 4
```

It seems that the program is waiting a string of 112 chars.

```
$ python -c 'print("a"*112)' | QEMU_LD_PREFIX=./microblaze-linux
↪ LD_LIBRARY_PATH=./microblaze-linux/lib qemu-microblaze ./small_hashes_anyways
small hashes anyways:
mismatch 1 wanted 1993550816 got 3904355907
```

A 112 chars string looks like a flag. Let's try to put the beginning of a flag:

```
$ python -c 'print("flag{yankee957187romeo3:"+"a"*88)' |
↪ QEMU_LD_PREFIX=./microblaze-linux LD_LIBRARY_PATH=./microblaze-linux/lib
↪ qemu-microblaze ./small_hashes_anyways
small hashes anyways:
mismatch 25 wanted 4293277456 got 770421485
```

Perfect, it seems that we could bruteforce each char one by one.

Running the python script we get:

```
$ python solver.py
sending > flag{yankee957187romeo3:0..............................................
↪ .......................................
sending > flag{yankee957187romeo3:1..............................................
↪ .......................................
sending > flag{yankee957187romeo3:2..............................................
↪ .......................................
sending > flag{yankee957187romeo3:3..............................................
↪ .......................................
sending > flag{yankee957187romeo3:4..............................................
↪ .......................................


...

sending > flag{yankee957187romeo3:GFrYdvdroY6qM_Cw0RxxUgIbYZ1g0AcfLzCi5GQeGj8LbjDx-
↪ vP073tareu4XrsXVDieStMacTyqefeHwnt956a.
sending > flag{yankee957187romeo3:GFrYdvdroY6qM_Cw0RxxUgIbYZ1g0AcfLzCi5GQeGj8LbjDx-
↪ vP073tareu4XrsXVDieStMacTyqefeHwnt956b.
sending > flag{yankee957187romeo3:GFrYdvdroY6qM_Cw0RxxUgIbYZ1g0AcfLzCi5GQeGj8LbjDx-
↪ vP073tareu4XrsXVDieStMacTyqefeHwnt956c.

flag > flag{yankee957187romeo3:GFrYdvdroY6qM_Cw0RxxUgIbYZ1g0AcfLzCi5GQeGj8LbjDx-vP
↪ 073tareu4XrsXVDieStMacTyqefeHwnt956c}
```

6 Rocinante Strikes Back

6.1 Once Unop a Djikstar

- **Category**: Rocinante Strikes Back
- **Points:** 131
- **Solves:** 27
- **Description:**

> Great, the fancy new Starlunk constellation won't route my packets anymore, right before our big trip.
>
> The StarlunkHacks forum thread had a sketchy looking binary. Might as well download that before we leave.
>
> Why did I think working on my honeymoon was a good idea…

> You'll need these files to solve the challenge.
>
> https://static.2022.hackasat.com/iprh5d6hcl0ijycqme1pqd25ynht

The remote server provides an additional description:

> You were on your way to your honeymoon in Bora Bora when your ship, ShippyMcShipFace, breaks down. You find yourself stranded in the middle of the Pacific Ocean.
>
> Thankfully you have just subscribed to the new global Starlunk network for a very affordable $110/month! Unfortunately, adversaries have corrupted the binary that is used to determine what string of satellites to route packets through within the Starlunk network. Because you have been spending countless hours on Youtube learning about the new network, you know that the nearest base station to you is in Honolulu, Hawaii. You also managed to find the corrupted binary on the internet before you left for your trip. You and all those aboard ShippyMcShipFace are counting on you to patch the binary so that you can uncover the route to send your packets in order to get help.
>
> Do you have what it takes to save yourself and those aboard ShippyMcShipFace?
>
> Please submit the resulting route from ShippyMcShipFace to Honolulu. Only include Satellite ID's with the "Starlunk-" omitted. For instance if the output from the corrected binary was:
>
> ShippyMcShipFace
>
> Starlunk-00-901

> Starlunk-06-22
>
> Starlunk-105-38
>
> Honolulu
>
> You would submit: “00-901, 06-22, 105-38” without the quotes. Your answer:

6.1.1 Write-up

Write-up by Solar Wine team

The challenge provides a x86-64 Rust program which is corrupted (some instructions were replaced with `NOP`), 3 CSV files and a shell script containing a command to run the program with the files:

```
./starlunk ShippyMcShipFace Honolulu gateways.csv sats.csv users.csv
```

Each file gives the distance between two nodes. `gateways.csv` contains distances from Honolulu:

```
Source,Dest,Range
Honolulu,Starlunk-21-15,1867.46987808192
Honolulu,Starlunk-21-16,1959.72070327412
Honolulu,Starlunk-22-15,1365.92203974943
...
```

`sats.csv` contains distances between satellites:

```
Source,Dest,Range
Starlunk-30-13,Starlunk-62-5,1112.67910099661
Starlunk-30-13,Starlunk-62-6,1697.15265130369
Starlunk-30-13,Starlunk-63-5,1370.11476857795
Starlunk-30-13,Starlunk-64-5,1787.84975181093
...
```

and `users.csv` contains distances from the `ShippyMcShipFace` user:

```
Source,Dest,Range
ShippyMcShipFace,Starlunk-21-18,1972.19478055826
ShippyMcShipFace,Starlunk-22-17,1814.54535536371
ShippyMcShipFace,Starlunk-22-18,1783.05769496665
...
```

At first, this seems easy: load all the CSV files, find the shortest path, and that's it. But the path was not considered as valid. What is the program `starlunk` actually computing?

It implements a function named `once_unop_a_dijkstar::once_unop_a_dijkstar::run` which parses the content of the 3 CSV files and record the distances between nodes (identified in structures `once_unop_a_dijkstar::Node`). However, the function which actually parses the distances, `parse_reader_file`, invokes two suspicious functions: `determine_target_type` and `get_target_traversal_cost`:

```
impl once_unop_a_dijkstar::once_unop_a_dijkstar {
    fn determine_target_type(target_name: &str) -> TargetType {
        let Some(value) = std::str::parse<i32>(target_name[target_name.len() - 1]) {
            match value % 3 {
                0 => TargetType::Starmander,
                1 => TargetType::Starmeleon,
                2 => TargetType::Starzard,
            }
        } else {
            TargetType::UserOrGatewayStation
        }
    }

    fn get_target_traversal_cost(t_distance: double, t_type: TargetType) -> double {
        t_distance * determine_type_weight(&t_type)
    }

    fn determine_type_weight(t_type: &TargetType) -> double {
        match t_type {
            TargetType::Starmander => 2.718,
            TargetType::Starmeleon => 3.141,
            TargetType::Starzard => 4.04,
            TargetType::UserOrGatewayStation => 1999.9,
        }
    }
}
```

So the distances are weighted according to the type of satellite which is used as target!

The shortest weighted path to Honolulu is then:

- `ShippyMcShipFace` to `Starlunk-63-6`: 654.877698384289 with Starmander weight 2.718
- `Starlunk-63-6` to `Starlunk-58-7`: 1888.77054209945 with Starmeleon weight 3.141
- `Starlunk-58-7` to `Starlunk-53-8`: 1340.68207660327 with Starzard weight 4.04
- `Starlunk-53-8` to `Starlunk-24-15`: 346.11386170069 with Starzard weight 4.04

- `Starlunk-24-15` to `'Honolulu`: 607.039251048805 with UserOrGatewayStation weight 1999.9

(Cumulated: 654.877698384289*2.718 + 1888.77054209945*3.141 + 1340.68207660327*4.04 + 346.113861700069*4.04 + 607.039251048805*1999.9 = 1228545.039620196)

The remote server accepted this path:

```
63-6,58-7,53-8,24-15
You saved those aboard ShippyMcShipFace! Here's your flag:
flag{bravo600611golf3:GPdEcditK-2M16EKs6KIe86404oEWCUnQ8xfuF09Qsrft2aNiK9AbsDVhTdw-
 ↪  -J0ax8dGyLZUUzhaS6EOp1mZwc}
```

6.2 Stars Above

- **Category**: Rocinante Strikes Back
- **Points:** 371
- **Solves:** 2
- **Description:**

> Starlunk has been nothing but trouble since I subscribed. Those hackers really did a number on things.
>
> Maybe we can force the system to route our traffic better. Work smarter not harder right?

6.2.1 Write-up

Write-up by Solar Wine team

The challenge provides a remote service which expects hexadecimal input:

- (the server sent) `Successfully connected to 'space'`
- (we sent) `?`
- (the server sent) `Encountered Non-Hex string, dropping...`
- (we sent) `00`
- (the server sent) `Couldn't decode message from hex...`

We are also provided with a Rust binary compiled for x86-64. This program connects to some server provided through the environment variable `SERVER` (`localhost:31337` by default) and exchanges messages using a function named `client::comms::CommManager::start_communication`.

It receives messages encoded in hexadecimal which are deserialized using the Bincode format specified on https://github.com/bincode-org/bincode/blob/trunk/docs/spec.md.

The received message is decoded in a structure named `Message`. A possible way to obtain the fields of `Message` consists in reverse-engineering the functions which displays the content of a `Message` instance:

```
void <messages::messages::Message_as_core::fmt::Debug>::fmt(long
 ↪ param_1,undefined8 param_2)
{
  long local_30;
  undefined local_28 [16];

  local_28 = core::fmt::Formatter::debug_struct(param_2,"Message",7);
  local_30 = param_1;
```

```
  core::fmt::builders::DebugStruct::field
            (local_28,"route",5,&local_30,&PTR_drop_in_place<&messages_route_Route>
             ↪  _00166e68);
  local_30 = param_1 + 0x20;
  core::fmt::builders::DebugStruct::field
            (local_28,"body",4,&local_30,&PTR_drop_in_place<&messages_messages_Mess
             ↪  ageBody>_00166e88
             );
  core::fmt::builders::DebugStruct::finish(local_28);
  return;
}
```

This function calls two functions to decode a field named `route` of type `messages::route::Route` and a field named `body` of type `messages::messages::MessageBody`. Repeating this step leads to decoding the following Rust structure:

```
struct Message {
    route: struct Route {
        src: u16,
        dst: Vec<u16>,
    }
    body: enum MessageBody {
        Hello: u16, // bincode tag 0
        NetworkCheckReq: u16, // bincode tag 1
        NetworkCheckAck: u16, // bincode tag 2
        SendFlag, // bincode tag 3
        Flag: String, // bincode tag 4
        DetourReq: (u16, u16), // bincode tag 5
    }
}
```

When running the program with `./challenge', "5", "32"`, it displays

```
Sending message Message { route: Route { src: 1, dst: [0] }, body: Hello(1) }
```

and sends 18 bytes to a server at `localhost:31337`. These bytes can be decoded (using Bincode format) in the following way:

```
0100                                    - src: u16 = 1
    0100000000000000                    - dst.len: u64 = 1
                    0000                - dst[0]: u16 = 0
```

```
                        00000000     - body.tag = 0 (Hello)
                                0100 - u16: 1
```

When we changed some hexadecimal bytes and wrote them on the remote server provided by the challenge, it displayed:

```
0100010000000000000000000000000200
Sending message Message { route: Route { src: 1, dst: [0] }, body: Hello(2) }

01000100000000000000000003000000
Sending message Message { route: Route { src: 1, dst: [0] }, body: SendFlag }
```

So we managed to send a command requesting to send the flag… but did not get any reply

Also, when changing the destination, we found out that most messages triggered a warning `Message dropped by 'physics'...`, except when sending a message to destination 12:

```
(sending) 0100010000000000000000c00010000000500
Message { route: Route { src: 99, dst: [99] }, body: NetworkCheckAck(99) }
Sending message Message { route: Route { src: 1, dst: [12] }, body:
 ↪  NetworkCheckReq(5) }
Message { route: Route { src: 30, dst: [5] }, body: NetworkCheckAck(12) }
```

We got a message from source 30 to destination 5.

At some point in reverse-engineering the Rust program, we stumbled upon a function named `routelib::get_route` which crafts the destination vector of a message being sent. It uses a table at address `0x55340`, holding 17 entries of 6 integers:

```
ID   Type   Neighbours
 0     2     0     0     0     0
 1     0    12     2     0    20
 2     0     1     3     0    20
 3     0     2     4     0    20
 4     0     3     5     0    30
 5     0     4     6     0    30
 6     0     5     7     0    30
 7     0     6     8     0    40
 8     0     7     9     0    40
 9     0     8    10     0    40
10     0     9    11     0    50
11     0    10    12     0    50
```

```
12      0     11      1      0     50
20      1      1      2      3      0
30      1      4      5      6      0
40      1      7      8      9      0
50      1     10     11     12      0
```

Here is a high-level description of what `routelib::get_route` does:

- There are 17 satellites: 0, 1, 2…, 12 and 20, 30, 40, 50.
- Satellites 1 to 12 are in a ring: each one can directly communicates with its neighbours.
- Satellites 20 to 50 are the gateways which can be each be used to reach 3 satellites:
  - 20 is connected to 1, 2, 3
  - 30 is connected to 4, 5, 6
  - 40 is connected to 7, 8, 9
  - 50 is connected to 10, 11, 12
- Satellite 0 connects together satellites 20 to 50.

For example, when satellite 5 wants to send a message to satellite 8, it crafts a route which goes through 30, 0, 40 and 8. It results in a message with `dst = [30, 0, 40, 8]`, received by 30. This satellite then forwards the message with `dst = [0, 40, 8]` to 0, which continues forwarding the message until 8 receives the message.

When sending the `SendFlag` message to satellite 5 (using `000002000000000000001e00050003000000`), the service displays:

```
Sending message Message { route: Route { src: 0, dst: [30, 5] }, body: SendFlag }
Sat_5: Starting Network Check...
Message { route: Route { src: 30, dst: [5] }, body: SendFlag }
Sat_5: Network Check Succeeded! Sending Flag to 8...
```

The satellite sends the flags to 8, and we are interested in the content of this message. How can we make the remote server also decode the message?

The program luckily includes a *Detour* feature which enables modifying the route that messages take. For example, if satellite 40 is configured with a detour `(8, 7)`, when it receives a message for 8, it will forward it to 7 instead. Such a detour can be set up by sending a `DetourReq` message.

Here is a program which sends `DetourReq` message so that when satellite 40 receives a message for 8, this message is forwarded to 7, 6, 5, 4, 3, 2, 1, 12, 11, 10, 9 and 8.

```python
import struct
from pwn import *

conn = remote("starsabove.satellitesabove.me", 5500)
line = conn.recvline()
assert line == b'Ticket please:\n', line
conn.sendline(
    b"ticket{lima322514xray3:GI8QCBCzjEwB3trVn6APiFUzoPDBt0GAaR_Qy9N5GN6JOfERY8HAH2HBN7Xa0jR81Q}")

line = conn.recvline()
print(f"[{datetime.datetime.now().strftime('%H:%M:%S')} CLIENT] {line!r}")
assert line == b"Successfully connected to 'space'\n", line

def send_msg(src, dst, msgtype, param):
    msg = struct.pack('<HQ', src, len(dst))
    for d in dst:
        msg += struct.pack('<H', d)
    msg += struct.pack('<I', msgtype) + param
    print(f"[CLIENT] ({src},{dst},{msgtype}:{param.hex()}) > {pkt.hex()}")
    conn.sendline(pkt.hex().encode("ascii"))

def send_hello(src, dst, number):
    send_msg(src, dst, 0, struct.pack("<H", number))

def send_net_check_req(src, dst, number):
    send_msg(src, dst, 1, struct.pack("<H", number))

def send_sendflag(src, dst):
    send_msg(src, dst, 3, b'')

def send_detourreq(src, dst, number1, number2):
    send_msg(src, dst, 5, struct.pack("<HH", number1, number2))

# Send messages from satellite 5 to retoute packets targeting 8
send_detourreq(5, [30, 0, 40], 8, 7)
send_detourreq(5, [30, 0, 40, 7], 8, 6)
send_detourreq(5, [6], 8, 5)
send_detourreq(5, [30, 5], 8, 4)
send_detourreq(5, [4], 8, 3)
send_detourreq(5, [30, 0, 20, 3], 8, 2)
send_detourreq(5, [30, 0, 20, 2], 8, 1)
send_detourreq(5, [30, 0, 20, 1], 8, 12)
send_detourreq(5, [30, 0, 50, 12], 8, 11)
send_detourreq(5, [30, 0, 50, 11], 8, 10)
send_detourreq(5, [30, 0, 50, 10], 8, 9)
```

```
send_sendflag(5, [30, 0, 20, 2])
```

When sending the `SendFlag` request, the following log appears:

```
Sending message Message { route: Route { src: 5, dst: [30, 0, 20, 2] }, body:
 ↪  SendFlag }
Sat_2: Starting Network Check...
Message { route: Route { src: 30, dst: [5] }, body: NetworkCheckReq(2) }
Sat_2: Network Check Failed! Goodbye...
Message { route: Route { src: 6, dst: [5, 8] }, body: NetworkCheckReq(2) }
Sat_2: Network Check Succeeded! Sending Flag to 8...
Message { route: Route { src: 6, dst: [5, 8] }, body:
 ↪  Flag("flag{lima322514xray3:GDa5BepzVbgOQPIVwGqm3E3oPh8ujJLiYjYckvLkmjJxHx4PoFf
 ↪  LUPyC_wvOXtXYk6gwPWRcsET8VDfvrmevRKY}")
 ↪  }
```

6.3 The Wrath of Khan

- **Category**: Rocinante Strikes Back
- **Points:** 234
- **Solves:** 10
- **Description:**

> You thought Khan was defeated and you were safe. Wrong! He messed with the hardware on the Genesis probe and now its going to crash. Can you patch the software so that it safely lands on Seti-Beta VII. Seti-Beta VII is an M class planet with:
>
> Radius: 13120 km
>
> Gravitational Constant: 2527105.1264 (km3/s2)
>
> You'll need these files to solve the challenge.
>
> - https://static.2022.hackasat.com/d85oddam8vndqco336q5gvnf0orh

6.3.1 Write-up

Write-up by Solar Wine team

The challenge provides an ARM64 firmware and a way to upload it to a remote server:

```
Ticket please:
Upload binary at: 3.222.113.106:19634

Simulation complete
Try again...
Telemetry available for  at 3.222.113.106:19634/tm
```

When we go on the telemetry, we see nice sinusoidal curves.

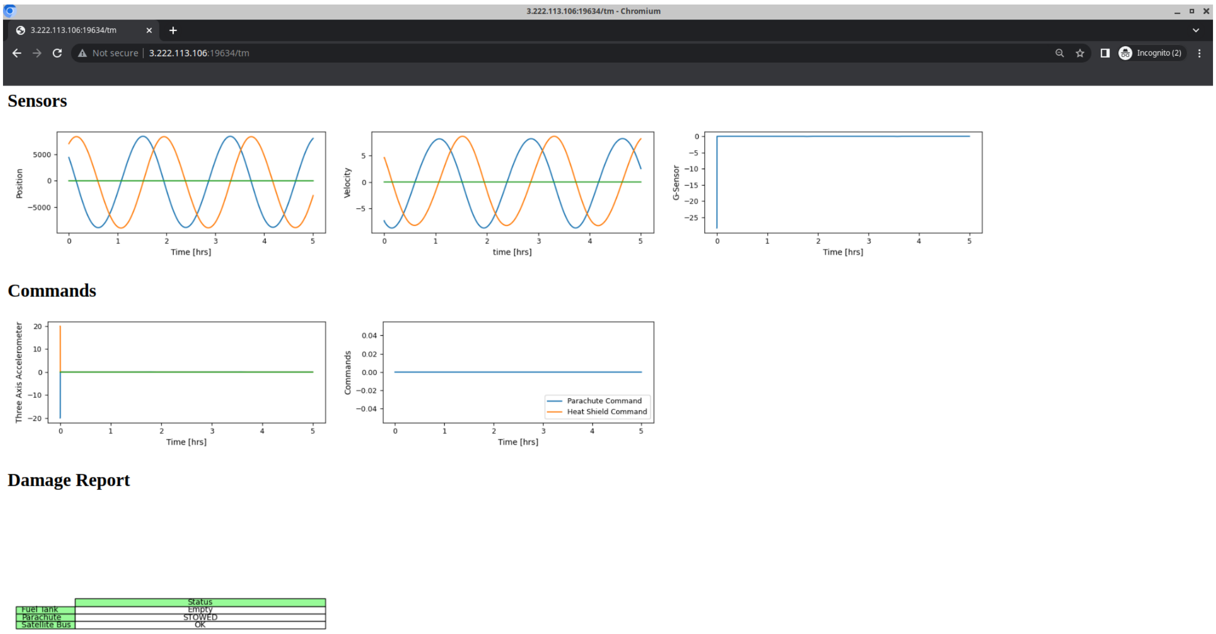

Figure 4: Telemetry with the given firmware

So it seems the probe is still orbiting around the planet and its fuel tank was emptied after trying to accelerate too much at the beginning of the simulation. Why did it behave like this?

The firmware is a C++ program which can be in 4 states, represented by C++ classes:

- `DeOrbit` : compute a deceleration vector to make the satellite lose altitude, until it reaches 100km above the radius (13120 km).
- `WaitCool` : wait for the velocity of the satellite to be below 1.0. It then triggers the parachute.
- `WaitChute` : wait for some sensor to be below 15.0. It then triggers the heat shield (according to the telemetry).
- `Touchdown` : stay at this state.

As the probe is decelerating too much, the issue could be in the first state. The deceleration vector is computed using the following algorithm:

- Read the position r from the sensors
- Read the velocity v from the sensors
- Compute $H = r \times v$ (this is a vector product)
- Compute $e = \frac{v \times H}{2527105.1264} - \frac{r}{|r|}$
- Compute $a = \frac{|H|^2}{2527105.1264}(1 - |e|^2)(1 - |e|)$
- Compute $\delta = a - (100 + 13120)$

- If $|\delta| < 1$, go to `WaitCool` state
- Feed δ into a PID controller which outputs a value $accel$
- Send an acceleration command according to the vector $-accel\frac{v}{|v|}$

What do these equations compute? The parameters of a Kepler orbit! H is the angular momentum, e the eccentricity vector of the conical section and a is something like a distance… $\frac{|H|^2}{2527105.1264}$ is the parameter p of the conical section, but multiplying it with $1-|e|^2$ or $1-|e|$ does not really seem to make sense: the minimal radius would be $\frac{p}{1+|e|}$, the maximal radius $\frac{p}{1-|e|}$ and the large radius of the ellipse $\frac{p}{1-|e|^2}$.

Anyway, as the eccentricy is small (the orbit seems to be quite circular in the telemetry), this does not seem important. A more important issue seems to be that the value given by the telemetry does not match the radius at all! Indeed the position oscillates between -9000 and 9000, which is far below the radius of the planet, 13120 km.

Moreover, using a quick approximation considering that the orbit is circular, with a radius of 9000 and a velocity of 9, the period should be:

$$T = \frac{2\pi 9000}{9} = 6283$$

The telemetry shows a period of 2 hours, which matches T in seconds.

Then, the gravitational constant of the observed orbit is:

$$\mu = |r||v|^2 = 9000 \times 9^2 = 729000$$

This is not 2527105.1264 km3/s2 at all. Or is it, using another unit?

Let's suppose that the probe is orbiting at 9000 units from the center of the planet, with a speed of 9 units per seconds. We know that the time scale seems to be correct thanks to the observed period of the orbit, 2 hours. The gravitational constant is 729000 unit3/s2 so there is an equation:

$$729000 \text{ unit3/s2} = 2527105.1264 \text{ km3/s2}$$

$$1 \text{ unit3} = \frac{2527105.1264}{729000} \text{ km3}$$

$$1 \text{ unit} = \sqrt[3]{\frac{2527105.1264}{729000}} \text{ km} = 1.51 \text{ km}$$

In the common unit systems, 1 mile is 1.60934 km. This seems to match the unit used here!

In short, the firmware was writing with kilometers and the probe reads its position and velocity using miles. There are two way to fix this issue: converting everything to kilometers, or everything to miles.

We decided to convert everything to miles, using the Gravitational Constant: $\frac{2527105.1264}{1.60934^3} = 606289.28285$ miles3/s2 and the radius $\frac{13120}{1.60934} = 8152.4103$ miles. In practice, we replaced the Gravitational Constant located at address 0xba28 and the instructions located at address 0x1c6c and 0x1c70 from:

```
00001c6c   00 00 d4 d2         mov        x0,#0xa00000000000
00001c70   20 19 e8 f2         movk       x0,#0x40c9, LSL #48
```

to

```
00001c6c   20 0d db d2         mov        x0,#0xd86900000000
00001c70   e0 17 e8 f2         movk       x0,#0x40bf, LSL #48
```

Using this modified firmware, the simulator reported:

```
Parachute damaged at 0.5208333333333334
You hit the planet and destroyed the probe at 28.752938935315463 m/s
Simulation complete
Try again...
```

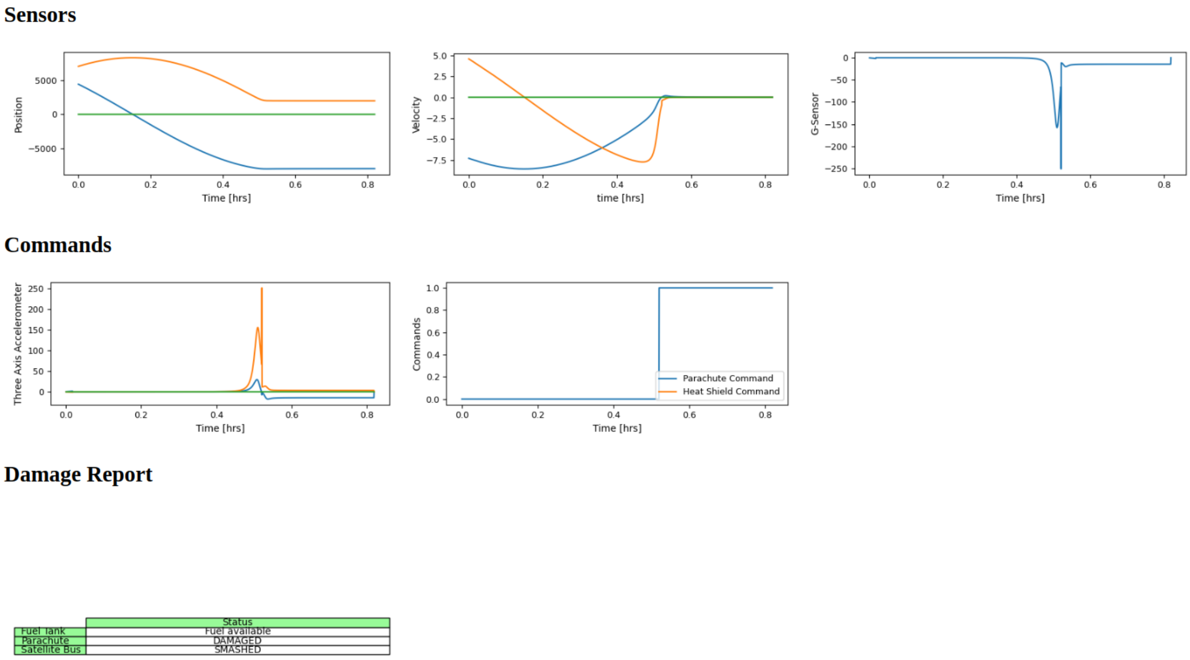

Figure 5: Result: damaged parachute

The telemetry curves clearly show that the satellite is no longer orbiting, but the parachute was triggered too early. Maybe the test "wait for the velocity to be below 1.0" in the `WaitCool` state should be updated as well? By modifying the instruction located at 0x792c to use 0.25 insted of 1.0 worked:

```
What a great landing
Simulation complete
Here is your flag:
flag{zulu930957foxtrot3:GD2pdM5-IYaeDN2DUe9nhkW-yb4izm3iDsKEa9HpyxShPY4pPmYhJolr9L
↪ H23VLWNKhQlDbsCi7g17FNHWX4OFg}

Congrats!
```