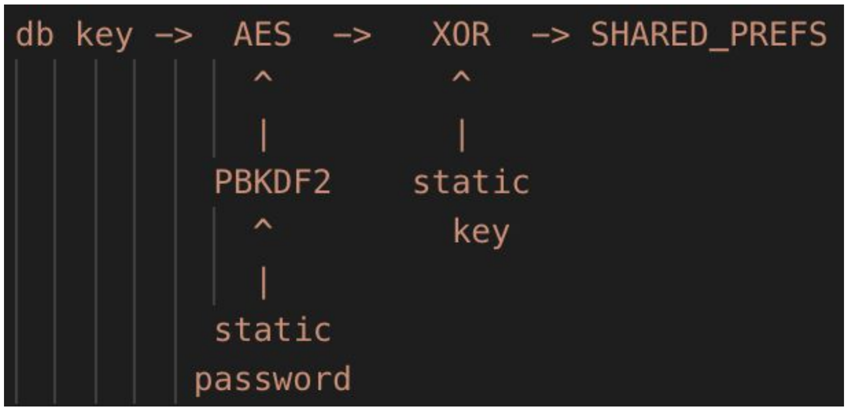
ROAD TO SPACE - MISC 400

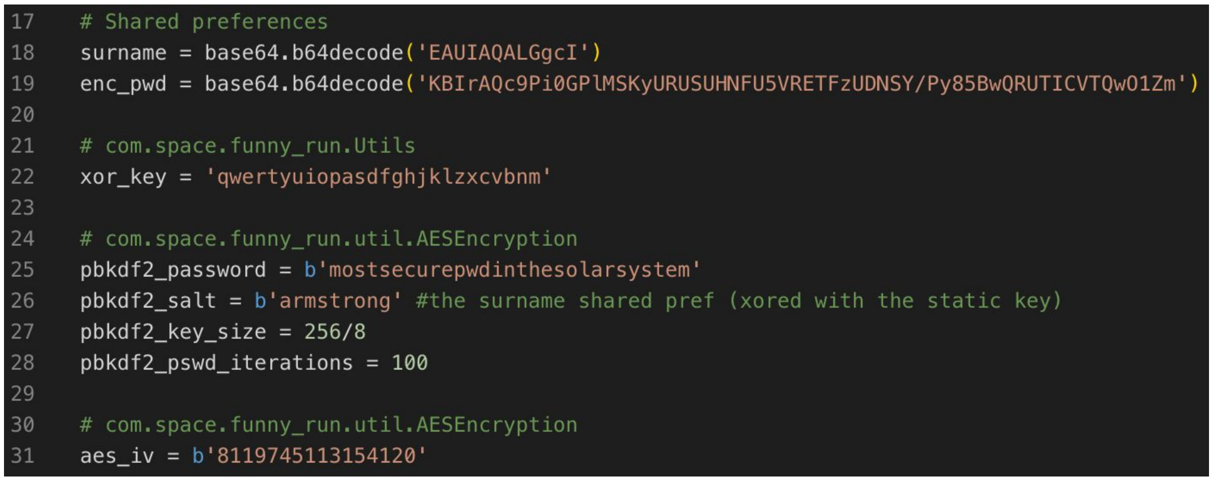
The kidnappers are sending encrypted GPS coordinates of Armstrong's location. Only the smartest minds can locate him. The situation is desperate: only R-boy can save the day!

In the app directory we found “com.space.funny_run-1”,“thanks” to multidex. After we decompiled all of it and analyzed the code we found that there was a database (also present in the dump in data directory) protected by a password (via SQLCipher 4).
That password was saved in the shared preferences protected by some layers of encryption as described in the low budget scheme below



```
db key -> AES  ->  XOR  -> SHARED_PREFS
           ^        ^
           |        |
         PBKDF2   static
           ^        key
           |
         static
        password
```

Also in the code we found all the parameters for the encryption



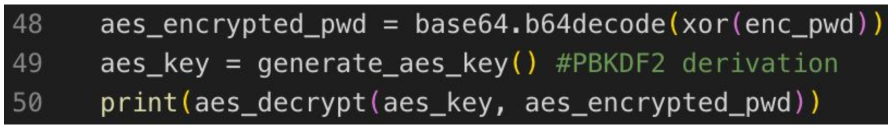
```python
# Shared preferences
surname = base64.b64decode('EAUIAQALGgcI')
enc_pwd = base64.b64decode('KBIrAQc9Pi0GPlMSKyURUSUHNFU5VRETFzUDNSY/Py85BwQRUTICVTQw01Zm')

# com.space.funny_run.Utils
xor_key = 'qwertyuiopasdfghjklzxcvbnm'

# com.space.funny_run.util.AESEncryption
pbkdf2_password = b'mostsecurepwdinthesolarsystem'
pbkdf2_salt = b'armstrong' #the surname shared pref (xored with the static key)
pbkdf2_key_size = 256/8
pbkdf2_pswd_iterations = 100

# com.space.funny_run.util.AESEncryption
aes_iv = b'8119745113154120'
```

We developed a script to get the database encryption key

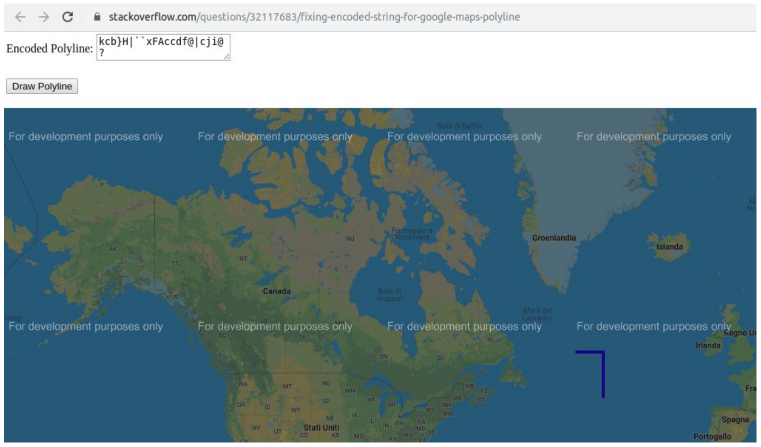


```python
aes_encrypted_pwd = base64.b64decode(xor(enc_pwd))
aes_key = generate_aes_key() #PBKDF2 derivation
print(aes_decrypt(aes_key, aes_encrypted_pwd))
```

In the end we opened the DB and exported the “Location” table in a JSON file in order to work faster with it

Database's name was a hint, we tried to print coordinates location and join them with Polylines by Google Maps but nothing appeared (bonus: all pins were in space agencies around the world).
The *CN_OPTIONS* column contains arrow images but they weren't useful.

Thanks to the hint "*Where you see trash, I see some lines*" we thought about using CN_IMAGE in *google.maps.geometry.encoding.decodePath* function.
In particular we used this snippet found on stackoverflow (2nd answer, 2nd snippet): https://stackoverflow.com/questions/32117683/fixing-encoded-string-for-google-maps-polyline
And finally some characters (from non-zero coordinates) appeared:


stackoverflow.com/questions/32117683/fixing-encoded-string-for-google-maps-polyline

Encoded Polyline:
```
kcb}H|``xFAccdf@|cji@
?
```

And manually, one by one we manage to get an hex string;

7b 46 4c 47 3a 79 30 75 47 30 54 74 68 65 50 30 49 4e 54 53 7d

Decode it and we got the flag:

{FLG:y0uG0TtheP0INTS}