

成员: NightlyAir

Reverse

Long long call

IDA 反汇编得到图 1，红框部分的逻辑是重复的猜测除了红框以外的部分才是真正的指令。

```
.text:0000000000001C43 sub_1C43 proc near ; CODE XREF: main+27p
.text:0000000000001C43 add     rsp, 8
.text:0000000000001C47 popfq
.text:0000000000001C48 mov     rsp, rsp
.text:0000000000001C48 pushfq
.text:0000000000001C4C call    sub_1C53
.text:0000000000001C51 leave
.text:0000000000001C52 retn
.text:0000000000001C52 sub_1C43 endp ; sp-analysis failed
.text:0000000000001C53
.text:0000000000001C53 ; ===== S U B R O U T I N E =====
.text:0000000000001C53 sub_1C53 proc near ; CODE XREF: sub_1C43+97p
.text:0000000000001C53 add     rsp, 8
.text:0000000000001C57 popfq
.text:0000000000001C57 sub     rsp, 40h
.text:0000000000001C58 pushfq
.text:0000000000001C5D call    sub_1C64
.text:0000000000001C62 leave
.text:0000000000001C63 sub_1C53 endp
.text:0000000000001C63
.text:0000000000001C64 ; ===== S U B R O U T I N E =====
.text:0000000000001C64 sub_1C64 proc near ; CODE XREF: sub_1C53+A7p
.text:0000000000001C64 add     rsp, 8
.text:0000000000001C68 popfq
.text:0000000000001C69 mov     rcx, rcx
.text:0000000000001C72 mov     [rbp-8], rcx
.text:0000000000001C76 pushfq
.text:0000000000001C77 call    sub_1C7E
.text:0000000000001C7C leave
.text:0000000000001C7D sub_1C64 endp ; sp-analysis failed
.text:0000000000001C7D
.text:0000000000001C7E
```

图 1，Long long call IDA 反汇编结果

写了个 python 脚本提取指令，如下：

```
1. import capstone
2. import lief
3.
4. def disasm(file):
5.     elf = lief.parse(file)
6.     md = capstone.Cs(capstone.CS_ARCH_X86, capstone.CS_MODE_64)
7.     text_seg = elf.get_section('.text')
8.     print(text_seg)
9.     data = text_seg.content
10.    main_addr = 0x15E7
11.    gen = md.disasm(data, text_seg.virtual_address)
12.    code_list = [insn for insn in gen]
13.    # skip to main_addr
14.    while 1:
```

```

15.     insn = code_list[0]
16.     if insn.address != main_addr:
17.         code_list.pop(0)
18.     else :
19.         break
20.     main_code = find_code(code_list)
21.     assert len(main_code[1]) == 0
22.     return main_code
23.
24. def find_code(code_list:list[capstone.CsInsn]):
25.     """
26.         返回一个元组，其中第一个元素为所有指令
27.         第二个元素为需要跳转的指令（默认情况下应为空）
28.         第三个元素为连接表（连接任意两个片段）
29.     """
30.     state = 0
31.     lst = []
32.     temp = []
33.     next_node = []
34.     connect_node = []
35.     connect_flag = False
36.     for (i,v) in enumerate(code_list):
37.
38.         if len(next_node) !=0:
39.             if next_node[-1][1].op_str == hex(v.address):
40.                 print("connect ",next_node[-1][0],"with", v)
41.                 connect_node.append((next_node.pop(),[]))
42.                 connect_flag = True
43.
44.         if v.mnemonic == 'add' and state==0:
45.             state=1
46.         elif v.mnemonic == 'popfq' and state==1:
47.             state=2
48.         elif state==2:
49.             temp.append(v)
50.             state =3
51.         elif state==3:
52.             if v.mnemonic == 'pushfq':
53.                 state = 4
54.             else:
55.                 # print("something wrong")
56.                 # print(v)
57.                 temp.append(v)
58.                 # state = 0

```

```

59.         elif state==4 and v.mnemonic == 'call':
60.             state = 5
61.             next_node.append((temp,v))
62.         elif state == 5 and v.mnemonic == 'leave':
63.             state = 6
64.         elif state == 6 and v.mnemonic == 'ret':
65.             lst.append(temp)
66.             if connect_flag:
67.                 connect_node[-1][1].extend(temp)
68.                 connect_flag = False
69.             temp = []
70.             state = 0
71.         else:
72.             print('unexpected!')
73.             print(v)
74.             state = 0
75.     return (lst,next_node,connect_node)
76.
77. def main():
78.
79.     main_code = disasm('./longlongcall')
80.     clousure = []
81.     try:
82.         while 1:
83.             p = main_code[2]
84.             n = p[0][1]
85.             lst = []
86.             lst.extend(p[0][0][0])
87.             lst.extend(n)
88.             for i in range(1,len(p)):
89.                 ...
90.                 if n == p[i][0][0]:
91.                     lst.extend(p[i][1])
92.                     n = p[i][1]
93.                     p[i] = []
94.             clousure.append(lst)
95.             p.pop(0)
96.             while p[0] == []:
97.                 p.pop(0)
98.     except IndexError:
99.         ...
100.    finally:
101.        ...
102.        for i in clousure:

```

```

103.         for j in i:
104.             print(j)
105.
106.
107.     main()

```

得到以下的汇编指令：

```

1.  Func_15ec:
2.  <CsInsn 0x15ec [55]: push rbp>
3.  <CsInsn 0x15fa [4889e5]: mov rbp, rsp>
4.  <CsInsn 0x160a [4881ecf0000000]: sub rsp, 0xf0>
5.  <CsInsn 0x161e [4889bd18ffffff]: mov qword ptr [rbp - 0xe8], rdi>
6.  <CsInsn 0x1632 [64488b042528000000]: mov rax, qword ptr fs:[0x28]
>
7.  <CsInsn 0x163b [488945f8]: mov qword ptr [rbp - 8], rax>
8.  <CsInsn 0x164c [31c0]: xor eax, eax>
9.  <CsInsn 0x165b [c7852cffffff00000000]: mov dword ptr [rbp - 0xd4]
, 0>
10. <CsInsn 0x1672 [e9a2020000]: jmp 0x1919>
11.
12. <CsInsn 0x1684 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
13. <CsInsn 0x1697 [4863d0]: movsxd rdx, eax>
14. <CsInsn 0x16a7 [488b8518ffffff]: mov rax, qword ptr [rbp - 0xe8]>
15. <CsInsn 0x16bb [4801d0]: add rax, rdx>
16. <CsInsn 0x16cb [0fb608]: movzx ecx, byte ptr [rax]>
17. <CsInsn 0x16db [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
18. <CsInsn 0x16ee [4898]: cdqe >
19. <CsInsn 0x16fd [488d5001]: lea rdx, [rax + 1]>
20. <CsInsn 0x170e [488b8518ffffff]: mov rax, qword ptr [rbp - 0xe8]>
21. <CsInsn 0x1722 [4801d0]: add rax, rdx>
22. <CsInsn 0x1732 [0fb600]: movzx eax, byte ptr [rax]>
23. <CsInsn 0x1742 [01c8]: add eax, ecx>
24. <CsInsn 0x1751 [88852bffffff]: mov byte ptr [rbp - 0xd5], al>
25. <CsInsn 0x1764 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
26. <CsInsn 0x1777 [4863d0]: movsxd rdx, eax>
27. <CsInsn 0x1787 [488b8518ffffff]: mov rax, qword ptr [rbp - 0xe8]>
28. <CsInsn 0x179b [4801d0]: add rax, rdx>
29. <CsInsn 0x17ab [0fb600]: movzx eax, byte ptr [rax]>
30. <CsInsn 0x17bb [8b952cffffff]: mov edx, dword ptr [rbp - 0xd4]>
31. <CsInsn 0x17ce [4863ca]: movsxd rcx, edx>
32. <CsInsn 0x17de [488b9518ffffff]: mov rdx, qword ptr [rbp - 0xe8]>
33. <CsInsn 0x17f2 [4801ca]: add rdx, rcx>
34. <CsInsn 0x1802 [32852bffffff]: xor al, byte ptr [rbp - 0xd5]>
35. <CsInsn 0x1815 [8802]: mov byte ptr [rdx], al>

```

```

36.
37.
38.  <CsInsn 0x1824 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
39.  <CsInsn 0x1837 [4898]: cdqe >
40.  <CsInsn 0x1846 [488d5001]: lea rdx, [rax + 1]>
41.  <CsInsn 0x1857 [488b8518ffffff]: mov rax, qword ptr [rbp - 0xe8]>
42.  <CsInsn 0x186b [4801d0]: add rax, rdx>
43.  <CsInsn 0x187b [0fb600]: movzx eax, byte ptr [rax]>
44.  <CsInsn 0x188b [8b952cffffff]: mov edx, dword ptr [rbp - 0xd4]>
45.  <CsInsn 0x189e [4863d2]: movsxd rdx, edx>
46.  <CsInsn 0x18ae [488d4a01]: lea rcx, [rdx + 1]>
47.  <CsInsn 0x18bf [488b9518ffffff]: mov rdx, qword ptr [rbp - 0xe8]>
48.  <CsInsn 0x18d3 [4801ca]: add rdx, rcx>
49.  <CsInsn 0x18e3 [32852bffffff]: xor al, byte ptr [rbp - 0xd5]>
50.  <CsInsn 0x18f6 [8802]: mov byte ptr [rdx], al>
51.
52.  <CsInsn 0x1905 [83852cffffff02]: add dword ptr [rbp - 0xd4], 2>
53.  <CsInsn 0x1919 [83bd2cffffff2b]: cmp dword ptr [rbp - 0xd4], 0x2b
>
54.  <CsInsn 0x1920 [0f8e5efdffff]: jle 0x1684>
55.  <CsInsn 0x1933 [90]: nop >
56.  <CsInsn 0x1941 [488b45f8]: mov rax, qword ptr [rbp - 8]>
57.  <CsInsn 0x1952 [64482b042528000000]: sub rax, qword ptr fs:[0x28]
>
58.  <CsInsn 0x195b [741f]: je 0x197c>
59.  <CsInsn 0x196a [e8f1f6ffff]: call __stack_chk_fail>
60.  <CsInsn 0x197c [c9]: leave >
61.  <CsInsn 0x198a [c3]: ret >
62.  Func 1998L=:
63.  <CsInsn 0x1998 [55]: push rbp>
64.  <CsInsn 0x19a6 [4889e5]: mov rbp, rsp>
65.  <CsInsn 0x19b6 [4881ecf00000000]: sub rsp, 0xf0>
66.  <CsInsn 0x19ca [4889bd18ffffff]: mov qword ptr [rbp - 0xe8], rdi>
67.  <CsInsn 0x19de [64488b042528000000]: mov rax, qword ptr fs:[0x28]
>
68.  <CsInsn 0x19e7 [488945f8]: mov qword ptr [rbp - 8], rax>
69.  <CsInsn 0x19f8 [31c0]: xor eax, eax>
70.  <CsInsn 0x1a07 [c7852cffffff00000000]: mov dword ptr [rbp - 0xd4]
, 0>
71.  <CsInsn 0x1a1e [e9a3010000]: jmp 0x1bc6>
72.  <CsInsn 0x1a30 [488d0501060000]: lea rax, [rip + 0x601]> //"check
ing"
73.  <CsInsn 0x1a44 [4889c7]: mov rdi, rax>
74.  <CsInsn 0x1a54 [e8e7f5ffff]: call puts>

```

```

75.  <CsInsn 0x1a66 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
76.  <CsInsn 0x1a79 [01c0]: add eax, eax>
77.  <CsInsn 0x1a88 [89c7]: mov edi, eax>
78.  <CsInsn 0x1a97 [e824f6ffff]: call sleep>
79.  <CsInsn 0x1aa9 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
80.  <CsInsn 0x1abc [4863d0]: movsxd rdx, eax>
81.  <CsInsn 0x1acc [488b8518ffffff]: mov rax, qword ptr [rbp - 0xe8]>
82.  <CsInsn 0x1ae0 [4801d0]: add rax, rdx>
83.  <CsInsn 0x1af0 [0fb610]: movzx edx, byte ptr [rax]>
84.  <CsInsn 0x1b00 [8b852cffffff]: mov eax, dword ptr [rbp - 0xd4]>
85.  <CsInsn 0x1b13 [4898]: cdqe >
86.  <CsInsn 0x1b22 [488d0d57250000]: lea rcx, [rip + 0x2557]>
87.  <CsInsn 0x1b36 [0fb60408]: movzx eax, byte ptr [rax + rcx]>
88.  <CsInsn 0x1b47 [38c2]: cmp dl, al>
89.  <CsInsn 0x1b49 [7467]: je 0x1bb2>
90.  <CsInsn 0x1b58 [488d05e5040000]: lea rax, [rip + 0x4e5]> //"Wrong"
91.  <CsInsn 0x1b6c [4889c7]: mov rdi, rax>
92.  <CsInsn 0x1b7c [e8bff4ffff]: call puts>
93.  <CsInsn 0x1b8e [bf01000000]: mov edi, 1>
94.  <CsInsn 0x1ba0 [e80bf5ffff]: call exit>
95.  <CsInsn 0x1bb2 [83852cffffff01]: add dword ptr [rbp - 0xd4], 1>
96.  <CsInsn 0x1bc6 [83bd2cffffff2b]: cmp dword ptr [rbp - 0xd4], 0x2b
>
97.  <CsInsn 0x1bcd [0f8e5dfeffff]: jle 0x1a30>
98.  <CsInsn 0x1be0 [488d0564040000]: lea rax, [rip + 0x464]> //"Right"
99.  <CsInsn 0x1bf4 [4889c7]: mov rdi, rax>
100. <CsInsn 0x1c04 [e837f4ffff]: call puts>
101. <CsInsn 0x1c16 [bf00000000]: mov edi, 0>
102. <CsInsn 0x1c28 [e883f4ffff]: call exit>
103. <CsInsn 0x1c3a [55]: push rbp>
104.
105. main:
106. <CsInsn 0x1c48 [4889e5]: mov rbp, rsp>
107. <CsInsn 0x1c58 [4883ec40]: sub rsp, 0x40>
108. <CsInsn 0x1c69 [64488b042528000000]: mov rax, qword ptr fs:[0x28]
>
109. <CsInsn 0x1c72 [488945f8]: mov qword ptr [rbp - 8], rax>
110. <CsInsn 0x1c83 [31c0]: xor eax, eax>
111. <CsInsn 0x1c92 [488d05b8030000]: lea rax, [rip + 0x3b8]> //"input
your flag:"
112. <CsInsn 0x1ca6 [4889c7]: mov rdi, rax>
113. <CsInsn 0x1cb6 [e885f3ffff]: call puts>
114. <CsInsn 0x1cc8 [488d45c0]: lea rax, [rbp - 0x40]> //字节数组
115. <CsInsn 0x1cd9 [4889c6]: mov rsi, rax>

```

```

116. <CsInsn 0x1ce9 [488d0572030000]: lea rax, [rip + 0x372]> // %44s
117. <CsInsn 0x1cfd [4889c7]: mov rdi, rax>
118. <CsInsn 0x1d0d [b800000000]: mov eax, 0>
119. <CsInsn 0x1d1f [e86cf3ffff]: call __isoc99_scanf>
120. <CsInsn 0x1d31 [488d052f030000]: lea rax, [rip + 0x32f]> // "ok, let's go"
121. <CsInsn 0x1d45 [4889c7]: mov rdi, rax>
122. <CsInsn 0x1d55 [e8e6f2ffff]: call puts>
123. <CsInsn 0x1d67 [488d45c0]: lea rax, [rbp - 0x40]>
124. <CsInsn 0x1d78 [4889c7]: mov rdi, rax>
125. <CsInsn 0x1d88 [e85ff8ffff]: call 0x15ec>
126. <CsInsn 0x1d9a [488d45c0]: lea rax, [rbp - 0x40]>
127. <CsInsn 0x1dab [4889c7]: mov rdi, rax>
128. <CsInsn 0x1dbb [e8d8fbffff]: call 0x1998>
129. <CsInsn 0x1dcd [b800000000]: mov eax, 0>
130. <CsInsn 0x1ddf [488b55f8]: mov rdx, qword ptr [rbp - 8]>
131. <CsInsn 0x1df0 [64482b142528000000]: sub rdx, qword ptr fs:[0x28]>
132. <CsInsn 0x1df9 [741f]: je 0x1e1a>
133. <CsInsn 0x1e08 [e853f2ffff]: call __stack_chk_fail>
134. <CsInsn 0x1e1a [c9]: leave >
135. <CsInsn 0x1e28 [c3]: ret >

```

经过分析 `func_15ec` 的逻辑为取两个字符的和，再将两个字符模上和。

写出爆破脚本：

```

1. lst = [
2.     0xBB, 0xBF, 0xB9, 0xBE, 0xC3, 0xCC, 0xCE, 0xDC, 0x9E, 0x8F,
3.     0x9D, 0x9B, 0xA7, 0x8C, 0xD7, 0x95, 0xB0, 0xAD, 0xBD, 0xB4,
4.     0x88, 0xAF, 0x92, 0xD0, 0xCF, 0xA1, 0xA3, 0x92, 0xB7, 0xB4,
5.     0xC9, 0x9E, 0x94, 0xA7, 0xAE, 0xF0, 0xA1, 0x99, 0xC0, 0xE3,
6.     0xB4, 0xB4, 0xBF, 0xE3
7. ]
8. lst = [[lst[i], lst[i+1]] for i in range(0, len(lst), 2)]
9. print(lst)
10. s = ''
11. for i in lst:
12.     try:
13.         for c in range(255):
14.             a = c^i[0]
15.             b = c^i[1]

```

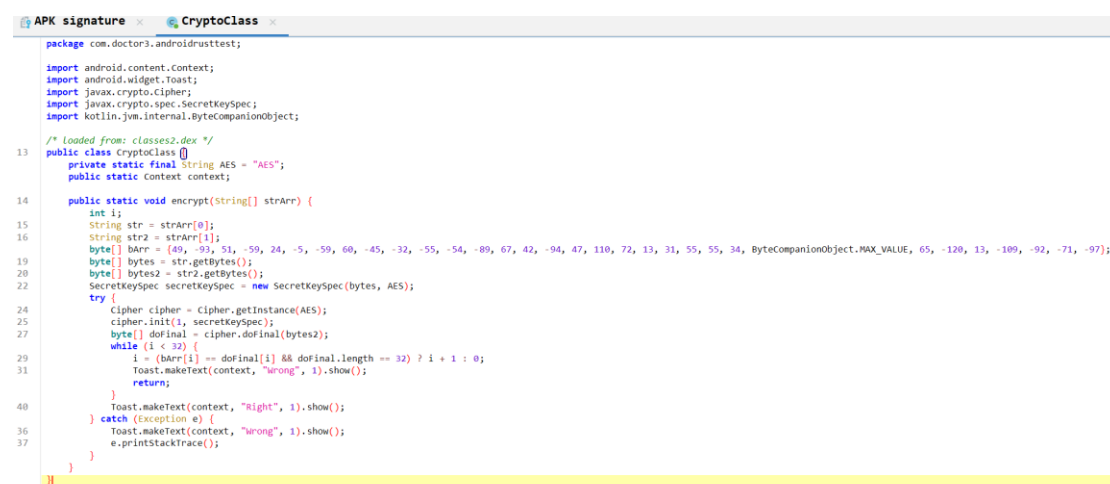
```

16.         if a+b == c:
17.             s+=chr(a)
18.             s+=chr(b)
19.             continue
20.     except ....:
21.         ...
22.
23.print(s)

```

RustedRobot

拖入 jadx 分析，发现涉及 Native 函数。结合实际测试，又发现在输入错误的 flag 时会有 Toast 打印信息，搜索对应的错误字符串得到了加密类。



```

package com.doctor3.androidrusttest;

import android.content.Context;
import android.widget.Toast;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import kotlin.jvm.internal.ByteCompanionObject;

/* loaded from: classes2.dex */
public class CryptoClass {
    private static final String AES = "AES";
    public static Context context;

    public static void encrypt(String[] strArr) {
        int i;
        String str = strArr[0];
        String str2 = strArr[1];
        byte[] bArr = {49, -53, 51, -59, 24, -5, -59, 60, -45, -32, -55, -54, -89, 67, 42, -94, 47, 110, 72, 13, 31, 55, 55, 34, ByteCompanionObject.MAX_VALUE, 65, -120, 13, -109, -92, -71, -97};
        byte[] bytes = str.getBytes();
        byte[] bytes2 = str2.getBytes();
        SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, AES);
        try {
            Cipher cipher = Cipher.getInstance(AES);
            cipher.init(1, secretKeySpec);
            byte[] doFinal = cipher.doFinal(bytes2);
            while (i < 32) {
                i = (bArr[i] == doFinal[i] && doFinal.length == 32) ? i + 1 : 0;
                Toast.makeText(context, "wrong", 1).show();
                return;
            }
            Toast.makeText(context, "right", 1).show();
        } catch (Exception e) {
            Toast.makeText(context, "wrong", 1).show();
            e.printStackTrace();
        }
    }
}

```

分析对应的 java 代码，可以见到这个函数会传入一个 String 数组，同时，搜索 encrypt 的函数调用，发现没有函数在 java 层调用它。

这时我想到了用 Frida 去 hook 得到 encrypt 函数的参数，由于我手头暂时没有 root 的设备，我想到了用 frida-gadget 注入在 apk 包中。

[apk.sh](https://github.com/ashley-nelson/apk.sh) 是一个用于快捷注入的工具。

自动化注入工具会遇到 ApplicationManifest.xml 无法编译的问题，手动修改下就好。（至少比手动修改 smali 代码好）

同时 jadx 也提供了复制 frida 片段的功能，（这点确实好评），需要注意的是 hook 时机可能过早导致没有获取到 CryptoClass。可以先 hook `dalvik.system.PathClassLoader` 确保对应的类已经加载。

```
CryptoClass.encrypt is called: strArr=btdfA2jeeljf.1bp,hbmgmojn  
CryptoClass.encrypt is called: strArr=btdfA2jeeljf.1bp,765432
```

Hook 的结果显示，第一个应该是 AES 的密钥，第二个参数则与输入的内容有关，经过简单的实验可以看出，处理方法就是将输入的内容倒转并将其对应的编码加 1。

需要注意的是 JAVA 的默认 AES 模式为 ECB。（我被一群乱码折磨过后才发现）

OLLessVM

这题能做出算法加密算法简单功不可没。

上来的入手点是通过动态调试发现加密后的数据与原来的数据有自反性，（ENIGAMA?），随即脑洞大开，将加密数据放入本应是输入 flag 的内存，让 enc 处理得到的就是 flag 了。

Web

Snooker King

在这题上我还真以为要打够 1145 次洞，对 cc 的函数乱搞了一通才

发现啥也没给我。

既然在代码里啥也没有，我就猜测 flag 可能是在场景文件中，也就是一大堆 json 中。

经过搜索确实是这样的。~~（为什么全局搜索里不能搜索网络流的请求）~~

"miniLCTF{U_ju3T_Hlt_1145_sN0Ok3r_Balls?!}","

Misc

Laughing-Knife-No-Running

你是会玩梗的，解题没啥好说的照着条件做就行了。~~（后来才发现原来地点是会变的）（在 AI 的基础上加了亿点点修改，不是原创我很抱歉）~~

```
1. import math
2.
3. earth_radius = 6378.137 # 地球半径（千米）
4. min_distance = 10 # 最小距离（千米）
5. max_speed = 40 # 最大速度（米/秒）
6. min_speed = 10 # 最小速度（米/秒）
7. min_time_step = 1 # 最小时间间隔（秒）
8. max_time_step = 10 # 最大时间间隔（秒）
9. max_displacement = 100 # 最大位移（米）
10.
11. speed = 35 # 速度（米/秒）
12. duration = 1.2 # 时长（秒）
13.
14.
15.
```

```
16.
17.import json
18.import time
19.import requests
20.
21.upload_url = 'http://127.0.0.1:13705/location'
22.status_url = 'http://127.0.0.1:13705/status'
23.restart_url = 'http://127.0.0.1:13705/restart'
24.checkpoints_url = 'http://127.0.0.1:13705/checkpoints'
25.
26.def generate_geo_data(waypoints):
27.    # 检查输入的航点是否有效
28.    if not waypoints or len(waypoints) < 2:
29.        raise ValueError("至少需要两个航点")
30.
31.    # 计算航点之间的总距离
32.    total_distance = 0
33.    for i in range(len(waypoints) - 1):
34.        total_distance += haversine_distance(waypoints[i], waypoints[i
+ 1])
35.
36.    # 计算每度对应的距离
37.    # speed = 20 # 速度 (米/秒)
38.    # duration = 4 # 时长 (秒)
39.    per_degree = duration * speed * 360 / (2*math.pi * earth_radius * 1
000)
40.
41.    # 扩展航点列表, 确保航点之间距离不小于最小距离
42.    k = 0
43.    while min_distance - k * total_distance > 0:
44.        k += 1
45.        waypoints.extend([waypoints[0], waypoints[1], waypoints[2]])
46.
47.    # 生成目标点列表
48.    target_points = [waypoints[0]]
49.    for i in range(len(waypoints) - 1):
50.        p1 = target_points[-1]
51.        p2 = waypoints[i + 1]
52.        lat1 = p1["lat"]
53.        lon1 = p1["lon"]
54.        lat2 = p2["lat"]
55.        lon2 = p2["lon"]
56.
57.        dlat = lat1 - lat2
```

```

58.         dlon = lon1 - lon2
59.
60.         # x = lat1
61.         # y = lon1
62.
63.         # 沿纬度方向移动
64.         while lat1 - lat2 > 0:
65.             lat1 -= per_degree
66.             target_points.append({"lat": lat1, "lon": lon1})
67.         while lat1 - lat2 < 0:
68.             lat1 += per_degree
69.             target_points.append({"lat": lat1, "lon": lon1})
70.
71.         # 沿经度方向移动
72.         while lon1 - lon2 > 0:
73.             lon1 -= per_degree
74.             target_points.append({"lat": lat1, "lon": lon1})
75.         while lon1 - lon2 < 0:
76.             lon1 += per_degree
77.             target_points.append({"lat": lat1, "lon": lon1})
78.         # i-=1
79.
80.     return target_points
81.
82.
83. # 计算两点之间的距离（海 versine 公式）
84. def haversine_distance(p1, p2):
85.     lat1 = math.radians(p1["lat"])
86.     lon1 = math.radians(p1["lon"])
87.     lat2 = math.radians(p2["lat"])
88.     lon2 = math.radians(p2["lon"])
89.
90.     dlon = lon2 - lon1
91.     dlat = lat2 - lat1
92.
93.     a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2) ** 2
94.     c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
95.
96.     return earth_radius * c
97.
98.
99. requests.get(restart_url)
100. _s = requests.get(checkpoints_url).json()['checkpoints']

```

```

101.
102.     waypoints = [
103.         _s[0],
104.         _s[1],
105.         _s[2]
106.     ]
107.
108.     # waypoints = [
109.         #     {"lat": 34.132899, "lon": 108.846993}, # 北操西侧入口
110.         #     {"lat": 34.130214, "lon": 108.836061}, # 丁香餐厅北侧
111.         #     {"lat": 34.134961, "lon": 108.843728}, # 北门
112.         # ]
113.     print(haversine_distance(waypoints[0], waypoints[1]))
114.     data = generate_geo_data(waypoints)
115.
116.     dis = 0
117.     for i in range(len(data)-1):
118.         now = haversine_distance(data[i], data[i+1])*1000
119.         dis += now
120.
121.         assert now < max_speed*duration
122.
123.     assert dis/len(data)/duration > min_speed
124.     print(dis)
125.
126.
127.     for i in data:
128.         requests.post(upload_url, data=json.dumps({"lat": i['lat'], "lon": i['lon']}))
129.         time.sleep(duration)
130.         print(requests.get(status_url).json())

```

Blockchain

dps_1ove

题目不难，主要原理就是减多了就炸子。

减一个 $(1 \ll 16) - 1$ 就好了。