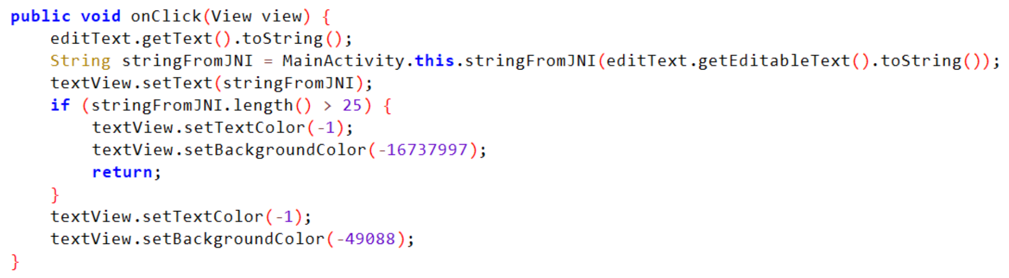
Lab8. Native Code Reversing

STU ID：19300240012
Your Flag：flag{26f9baf72c25abc95579b832645099cb}

Analysis Process Breakdown:

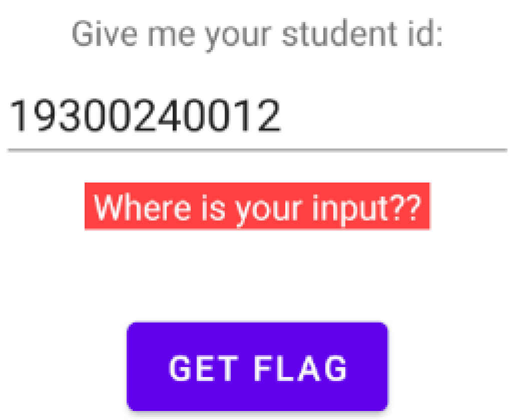
1.查看 java 代码



```java
public void onClick(View view) {
    editText.getText().toString();
    String stringFromJNI = MainActivity.this.stringFromJNI(editText.getEditableText().toString());
    textView.setText(stringFromJNI);
    if (stringFromJNI.length() > 25) {
        textView.setTextColor(-1);
        textView.setBackgroundColor(-16737997);
        return;
    }
    textView.setTextColor(-1);
    textView.setBackgroundColor(-49088);
}
```

先查看 java 代码，就是调用了 native 函数，输入一个字符串，放回一个字符串。

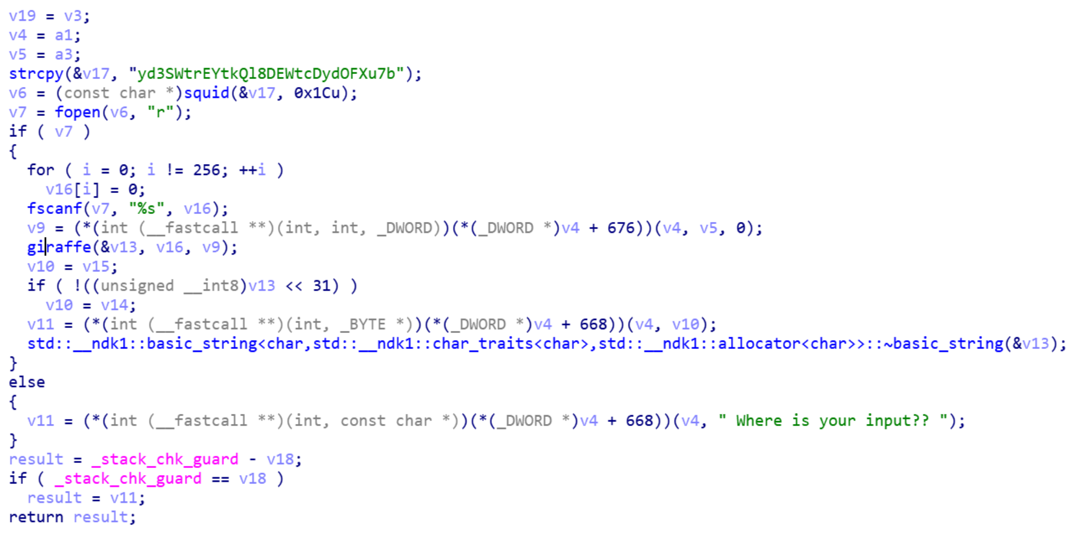
先试一下：



到这里进行不下去了，需要分析 native 代码

2.对 native 代码分析

利用 IDA 找到了对应的 native 函数



```
v19 = v3;
v4 = a1;
v5 = a3;
strcpy(&v17, "yd3SWtrEYtkQl8DEWtcDydOFXu7b");
v6 = (const char *)squid(&v17, 0x1Cu);
v7 = fopen(v6, "r");
if ( v7 )
{
  for ( i = 0; i != 256; ++i )
    v16[i] = 0;
  fscanf(v7, "%s", v16);
  v9 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v4 + 676))(v4, v5, 0);
  giraffe(&v13, v16, v9);
  v10 = v15;
  if ( !((unsigned __int8)v13 << 31) )
    v10 = v14;
  v11 = (*(int (__fastcall **)(int, _BYTE *))(*(_DWORD *)v4 + 668))(v4, v10);
  std::__ndk1::basic_string<char,std::__ndk1::char_traits<char>,std::__ndk1::allocator<char>>::~basic_string(&v13);
}
else
{
  v11 = (*(int (__fastcall **)(int, const char *))(*(_DWORD *)v4 + 668))(v4, " Where is your input?? ");
}
result = _stack_chk_guard - v18;
if ( _stack_chk_guard == v18 )
  result = v11;
return result;
```

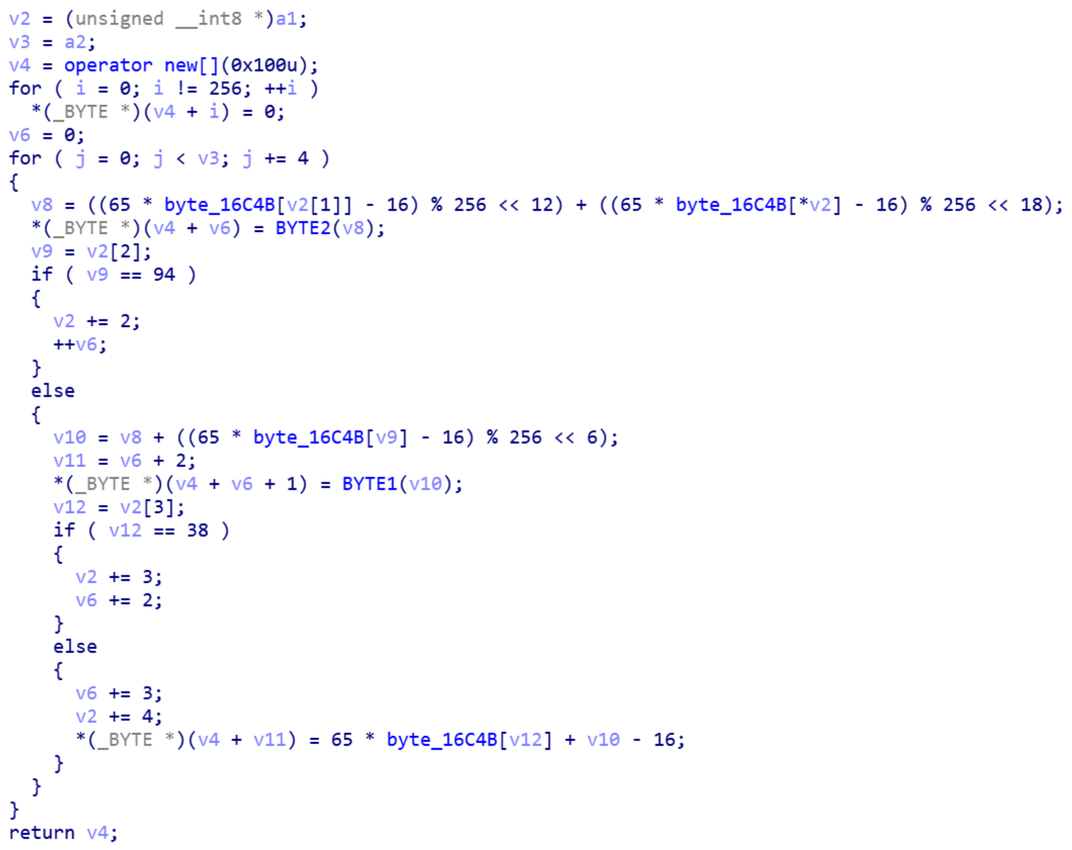
3.对 native 函数的理解

Native 函数主要进行了两件事，（1）把已传加密的字符串解密，根据解密的值打开一个文件，如果文件不存在输出" Where is your input?? " （2）对文件的内容进行判断，如果符合返回 flag，否则输出" Wrong Input :("

（1）解密函数 squid

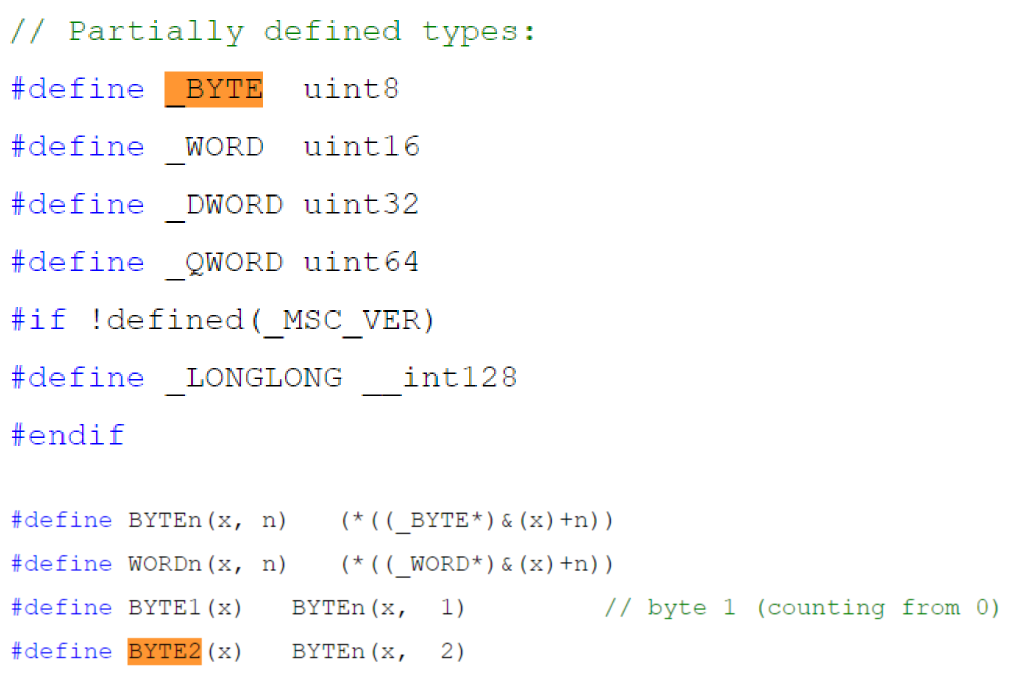
加密的字符串为：

yd3SWtrEYtkQl8DEWtcDydOFXu7b



```
v2 = (unsigned __int8 *)a1;
v3 = a2;
v4 = operator new[](0x100u);
for ( i = 0; i != 256; ++i )
  *(_BYTE *)(v4 + i) = 0;
v6 = 0;
for ( j = 0; j < v3; j += 4 )
{
  v8 = ((65 * byte_16C4B[v2[1]] - 16) % 256 << 12) + ((65 * byte_16C4B[*v2] - 16) % 256 << 18);
  *(_BYTE *)(v4 + v6) = BYTE2(v8);
  v9 = v2[2];
  if ( v9 == 94 )
  {
    v2 += 2;
    ++v6;
  }
  else
  {
    v10 = v8 + ((65 * byte_16C4B[v9] - 16) % 256 << 6);
    v11 = v6 + 2;
    *(_BYTE *)(v4 + v6 + 1) = BYTE1(v10);
    v12 = v2[3];
    if ( v12 == 38 )
    {
      v2 += 3;
      v6 += 2;
    }
    else
    {
      v6 += 3;
      v2 += 4;
      *(_BYTE *)(v4 + v11) = 65 * byte_16C4B[v12] + v10 - 16;
    }
  }
}
return v4;
```

对一些 c 中未定义的量，上网查到是 ida 的宏定义：



```
// Partially defined types:
#define _BYTE  uint8
#define _WORD  uint16
#define _DWORD uint32
#define _QWORD uint64
#if !defined(_MSC_VER)
#define _LONGLONG __int128
#endif

#define BYTEn(x, n)   (*((_BYTE*)&(x)+n))
#define WORDn(x, n)   (*((_WORD*)&(x)+n))
#define BYTE1(x)   BYTEn(x,  1)         // byte 1 (counting from 0)
#define BYTE2(x)   BYTEn(x,  2)
```

根据网上查找到的宏定义，补全这个函数的代码，同时编写 main 函数，运行后

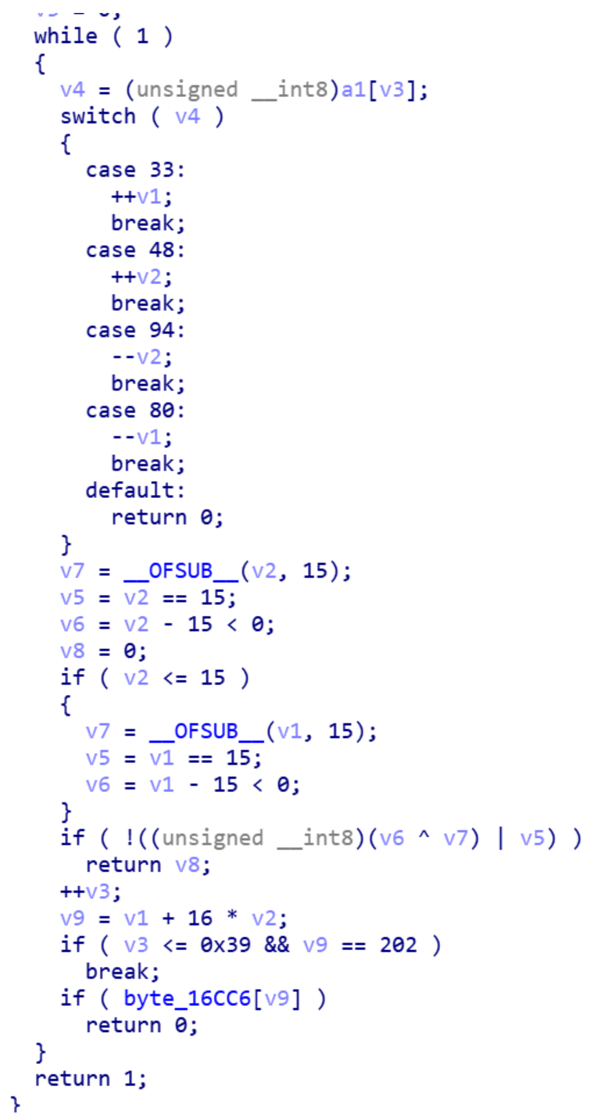
得到：

$ada/\ocQl/dmp)np5t

有点像，但并不是具体的路径，反复研究后，发现要修改一些变量的类型，将一些变量的类型由 int 改为 unsigned int，对字符串解密得到：

/data/local/tmp/input

（2）文件内容判断函数 giraffe 中的 panda



```
  while ( 1 )
  {
    v4 = (unsigned __int8)a1[v3];
    switch ( v4 )
    {
      case 33:
        ++v1;
        break;
      case 48:
        ++v2;
        break;
      case 94:
        --v2;
        break;
      case 80:
        --v1;
        break;
      default:
        return 0;
    }
    v7 = __OFSUB__(v2, 15);
    v5 = v2 == 15;
    v6 = v2 - 15 < 0;
    v8 = 0;
    if ( v2 <= 15 )
    {
      v7 = __OFSUB__(v1, 15);
      v5 = v1 == 15;
      v6 = v1 - 15 < 0;
    }
    if ( !((unsigned __int8)(v6 ^ v7) | v5) )
      return v8;
    ++v3;
    v9 = v1 + 16 * v2;
    if ( v3 <= 0x39 && v9 == 202 )
      break;
    if ( byte_16CC6[v9] )
      return 0;
  }
  return 1;
}
```

__OFSUB__(x,y) 表示 x-y 是否溢出,溢出返回 1,没有溢出返回 0

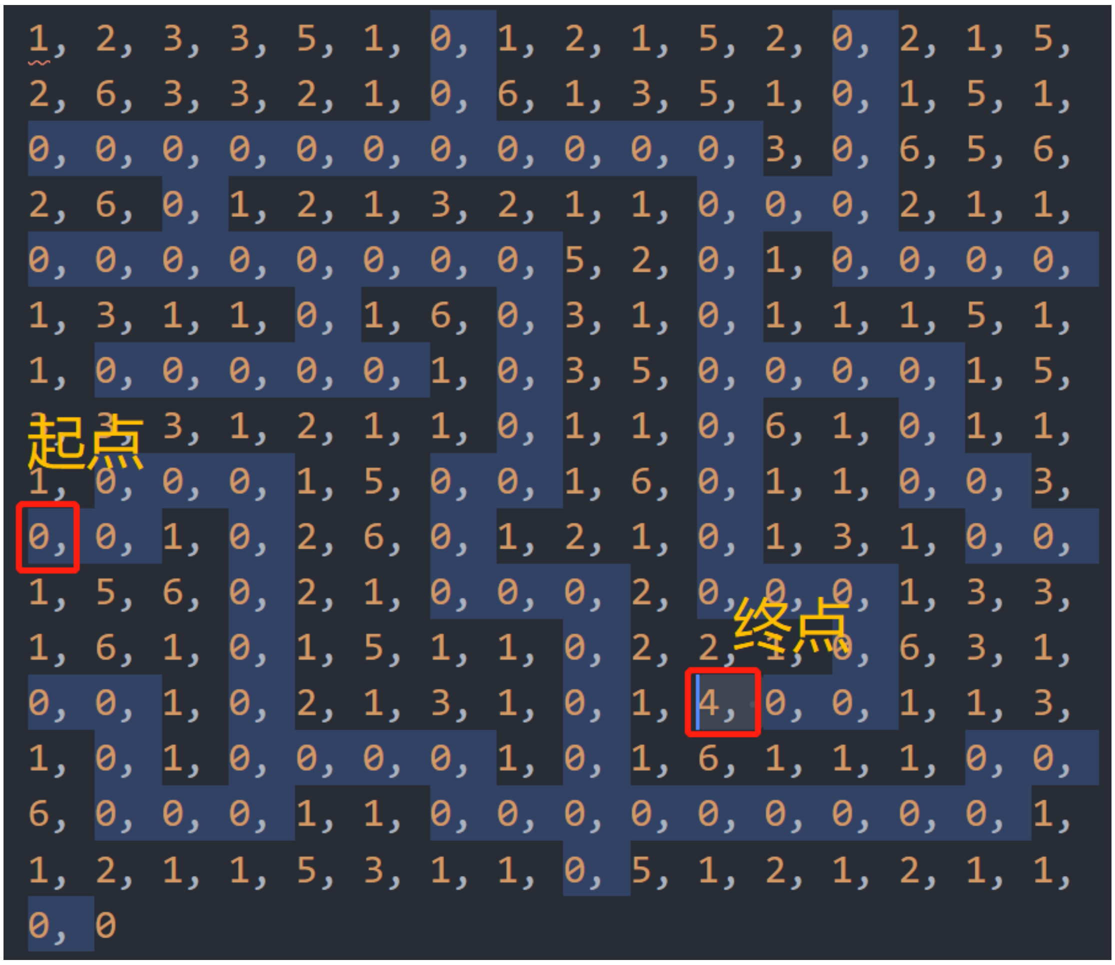
这里面同样有 ida 的宏定义，函数的逻辑可以认为是走迷宫，在矩阵中 0 表示可以通过，有数值表示为墙，从起点开始要到达终点，输入不同的值有不同含义：

!：向右

0：向下

^：向上

P：向左



```
1, 2, 3, 3, 5, 1, 0, 1, 2, 1, 5, 2, 0, 2, 1, 5,
2, 6, 3, 3, 2, 1, 0, 6, 1, 3, 5, 1, 0, 1, 5, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 6, 5, 6,
2, 6, 0, 1, 2, 1, 3, 2, 1, 1, 0, 0, 0, 2, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 5, 2, 0, 1, 0, 0, 0, 0,
1, 3, 1, 1, 0, 1, 6, 0, 3, 1, 0, 1, 1, 1, 5, 1,
1, 0, 0, 0, 0, 0, 1, 0, 3, 5, 0, 0, 0, 0, 1, 5,
3, 3, 3, 1, 2, 1, 1, 0, 1, 1, 0, 6, 1, 0, 1, 1,
1, 0, 0, 0, 1, 5, 0, 0, 1, 6, 0, 1, 1, 0, 0, 3,
0, 0, 1, 0, 2, 6, 0, 1, 2, 1, 0, 1, 3, 1, 0, 0,
1, 5, 6, 0, 2, 1, 0, 0, 0, 2, 0, 0, 0, 1, 3, 3,
1, 6, 1, 0, 1, 5, 1, 1, 0, 2, 2, 1, 0, 6, 3, 1,
0, 0, 1, 0, 2, 1, 3, 1, 0, 1, 4, 0, 0, 1, 1, 3,
1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 6, 1, 1, 1, 0, 0,
6, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 2, 1, 1, 5, 3, 1, 1, 0, 5, 1, 2, 1, 2, 1, 1,
0, 0
```

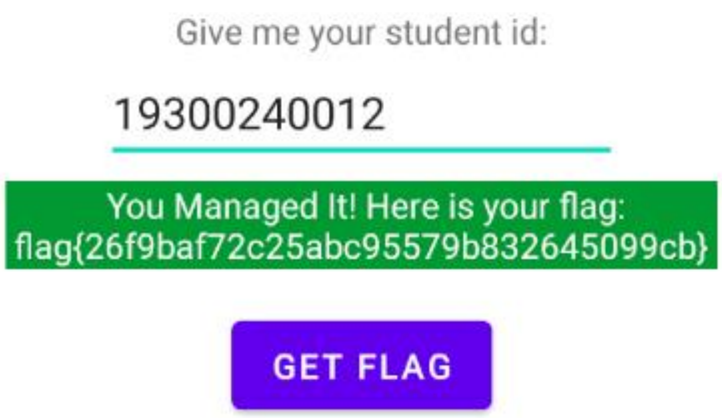
根据上面的推断得出文件的内容：

!^!!00000!!!0!!^^^^PP^^!^^^^PPPPP^^!!!!!!!00000000!!00PP

4.获得 flag

将文件 input

adb push 到/data/local/tmp/下，再点 get flag 获得 flag：



Give me your student id:

19300240012

You Managed It! Here is your flag:
flag{26f9baf72c25abc95579b832645099cb}

26f9baf72c25abc95579b832645099cb