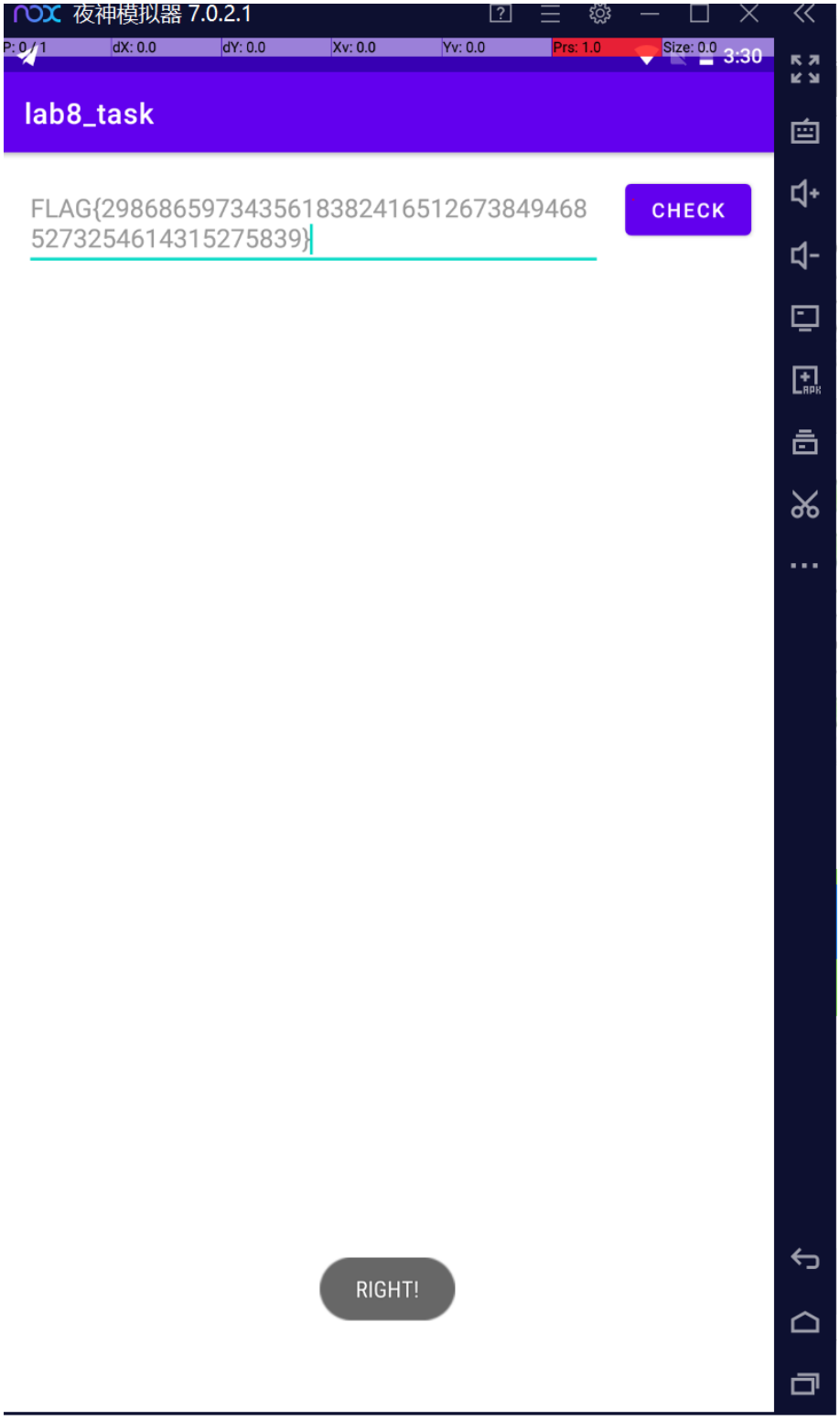
Lab8. Native Code Reversing

STU ID：20307130350

Your Flag：FLAG{298686597343561838241651267384946852732546143152758 39}



Analysis Process Breakdown:

长度=59

1. 用 jeb 打开 lab8_task.apk,知道 mainActivity，根据输入可以定位出输入的是参数是 v0，判断函数是 check(v0),所以参数是 String。

```
    }

    @Override  // androidx.fragment.app.FragmentActivity
    protected void onCreate(Bundle arg3) {
        super.onCreate(arg3);
        this.setContentView(0x7F0B001C);  // layout:activity_main
        TextView v3 = (TextView)this.findViewById(0x7F080005);  // id:Flag
        ((Button)this.findViewById(0x7F080064)).setOnClickListener(new View.OnClickListener() {  // id:button_
            @Override  // android.view.View$OnClickListener
            public void onClick(View arg3) {
                String v0 = v3.getText().toString();
                if(MainActivity.this.Check(v0)) {
                    Toast.makeText(MainActivity.this.getApplicationContext(), "RIGHT!", 0).show();
                    return;
                }

                Toast.makeText(MainActivity.this.getApplicationContext(), "WRONG!", 0).show();
            }
        });
    }
}
```

2. 用 IDA_pro 打开.so 文件，找到 check 函数，先修改传入的参数类型

```
t __cdecl Java_com_example_lab8_1task_MainActivity_Check(JNIEnv *a1, jobject a2, jstring a3)
```

3. 可以知道该字符串长度为 59，并赋值给了 v3，并且满足 FLAG{…(53)…}的形式。

```
v20 = __readgsdword(0x14u);
if ( (*a1)->GetStringUTFLength(a1, a3) == 59 )
{
  v3 = (*a1)->GetStringUTFChars(a1, a3, 0);
  if ( *v3 == 70 && v3[1] == 76 && v3[2] == 65 && v3[3] == 71 && v3[4] == 123 && v3[58] == 125 )
  {
    v4 = -53;
    while ( (unsigned __int8)(v3[v4 + 58] - 48) <= 9u )
    {
```

4. 因为长度 53 的字符数组赋值给了 dest 数组，所以将原先的 dest 数组的长度 16 修改成 53.报错。再次修改回来，继续往下看代码。

```
*(_OWORD *)dest = 0LL;
strncpy(dest, v3 + 5, 0x35u);
```

```
// local variable allocation has failed, the output may be wrong!
int __cdecl Java_com_example_lab8_1task_MainActivity_Check(JNIEnv *a1, jobje
{
  const char *v3; // eax
  int v4; // ecx
  int v5; // eax
  int v6; // ecx
  char v8; // dl
  char v9[16]; // [esp+0h] [ebp-BCh] BYREF
  __int128 v10; // [esp+10h] [ebp-ACh]
  __int128 v11; // [esp+20h] [ebp-9Ch]
  __int128 v12; // [esp+30h] [ebp-8Ch]
  __int128 v13; // [esp+40h] [ebp-7Ch]
  char v14; // [esp+50h] [ebp-6Ch]
  _BYTE v15[15]; // [esp+51h] [ebp-6Bh]
  char dest[53]; // [esp+60h] [ebp-5Ch] OVERLAPPED BYREF
  unsigned int v17; // [esp+A8h] [ebp-14h]
```

5. 直接定位到 return 处，进入调用函数，阅读代码，发现结果和 v9 相关，但是 v9 在 check 函数中“似乎”没有改变，因此 bbb(v9)无论如何返回都是 1。在 app 中随意输入 FLAG{1111…(53 个 1)…1111}，得到的结果是 wrong。因此推测在 check 函数中其实已经修改了 v9。

```
        v10 = 0LL;
        *(_OWORD *)v9 = 0LL;
        v14 = 0;
        v5 = 0;
        v6 = -81;
        do
        {
          v8 = *((_BYTE *)&ccc + v6 + 81);
          if ( !v8 )
            v8 = dest[v5++] - 48;
          v15[v6++] = v8;
        }
        while ( v6 );
        return bbb(v9);
      }
    }
  }
```

6. 考虑到 check 方法中对 v15 有所赋值，因此查看 v15 和 v9 的地址，发现二者的起始地址相差正好是 81，所以实际上是在给 v9 赋值。

```
-000000BC ; D/A/*    : change type (data/ascii/array)
-000000BC ; N        : rename
-000000BC ; U        : undefine
-000000BC ; Use data definition commands to create local va
-000000BC ; Two special fields " r" and " s" represent retu
-000000BC ; Frame size: BC; Saved regs: 4; Purge: 0
-000000BC ;
-000000BC
-000000BC var_BC          db 16 dup(?)
-000000AC anonymous_0     xmmword ?
-0000009C anonymous_1     xmmword ?
-0000008C anonymous_2     xmmword ?
-0000007C anonymous_3     xmmword ?
-0000006C anonymous_4     db ?
-0000006B                 db ? ; undefined
-0000006A                 db ? ; undefined
-00000069                 db ? ; undefined
-00000068                 db ? ; undefined
-00000067                 db ? ; undefined
-00000066                 db ? ; undefined
-00000065                 db ? ; undefined
-00000064                 db ? ; undefined
```

7. 由 do-while 循环可知，ccc 和 v15，v9 都是长度为 81 的 char[]，修改三者的变量类型，发现前面无法修改的 dest 也可以成功修改。

```
.data:00002000 ccc             db 0, 0, 1, 4, 3, 0, 7, 5, 0, 0, 0, 0, 0, 1, 0, 0, 4, 2, 0, 7, 0, 0, 0, 2, 9, 0, 0, 7, 0, 0, 0, 0,
.data:00002000                                 ; DATA XREF: LOAD:00000240↑o
```

```
{
  const char *v3; // eax
  int v4; // ecx
  int v5; // eax
  int v6; // ecx
  char v8; // dl
  char v9[81]; // [esp+0h] [ebp-BCh] BYREF
  _BYTE v10[81]; // [esp+51h] [ebp-6Bh] BYREF
  unsigned int v11; // [esp+A8h] [ebp-14h]

```

```
{
  v4 = -53;
  while ( (v3[v4 + 58] - 48) <= 9u )
  {
    if ( !++v4 )
    {
      *&v10[63] = 0LL;
      *&v10[47] = 0LL;
      *&v10[31] = 0LL;
      *&v10[15] = 0LL;
      strncpy(&v10[15], v3 + 5, 53u);
      *&v9[64] = 0LL;
      *&v9[48] = 0LL;
      *&v9[32] = 0LL;
      *&v9[16] = 0LL;
      *v9 = 0LL;
      v9[80] = 0;
      v5 = 0;
      v6 = -81;
      do
      {
        v8 = ccc[v6 + 81];
        if ( !v8 )
          v8 = v10[v5++ + 15] - 48;
        v10[v6++] = v8;
      }
      while ( v6 );
      return bbb(v9);
    }
```

8. 注意到当 v8==0 时，dest[i]会赋值给 v9，因此推测 ccc 中正好有 53 个 0，核对发现确实如此。可见前面的推理正确。

9. 进入 bbb 函数，考虑到 a1 的值都是 1~9，注意到 return 0 的条件语句，发现和 v32 和下一个地址的值有关。根据其原有的值，可见第一个循环不能满足任何情况。

```
LOAD:0000001C                   dd 34h
LOAD:00000020                   dd 1224h
LOAD:00000024                   dd 0
LOAD:00000028                   dw 34h
00000010  03 00 03 00 01 00 00 00  00 00 00 00
00000020  24 12 00 00 00 00 00 00  34 00 20 00
00000030  19 00 18 00 06 00 00 00  34 00 00 00
```

10. 关注赋值语句，注意到 v31 是 char 类型，其地址为 0000021，在 int v32 为 0000020，又

a1[81]都是 1~9，所以和 00020-00028 地址有关。

```
    *(&v31 + v2) = 1;
    v3 = a1[v1 + 82];
-00000029                   db ? ; undefined
-00000028                   db ? ; undefined
-00000027                   db ? ; undefined
-00000026                   db ? ; undefined
-00000025                   db ? ; undefined
-00000024                   db ? ; undefined
-00000023                   db ? ; undefined
-00000022                   db ? ; undefined
-00000021                   db ? ; undefined
-00000020                   db ? ; undefined
-0000001F                   db ? ; undefined
```

11. 快速浏览 bbb 方法的算法，发现第一个大循环运用到 v9（即 a1）下标的顺序是 0-1-2-…-8，9~17，….. 72~80，共 9 个；第二个大循环是 0-9-18-….-72, 1-10-….-73,….8-17-….-80，共 9 个；第三个循环是 0-1-2-9-10-11-18-19-21，3-4-5-12-13-14-22-23-24.……共 9 个。结合其 return 0 的 if 语句，可知这是个 9×9 的数独算法。而我需要做的就是根据 Ccc[81]已经提供的 28 个数字推导出另外的 53 个，这 53 即为 dest –{48，….}。
12. 将 ccc 已知的数字填入数独表格

		1	4	3		7	5	
				1			4	2
	7				2	9		
7					9			
	5						9	
			1					3
		9	7				8	
6	8			9				
	1	7		2	6	4		

解出

2	9	1	4	3	8	7	5	6
8	6	5	9	1	7	3	4	2
4	7	3	5	6	2	9	1	8
7	3	8	2	4	9	1	6	5
1	5	2	6	7	3	8	9	4
9	4	6	1	8	5	2	7	3
3	2	9	7	5	4	6	8	1
6	8	4	3	9	1	5	2	7
5	1	7	8	2	6	4	3	9

13. 用代码反解出 dest[53]，result=FLAG{dest}。
即 **FLAG{298686597343561838241651267384946852732546143152758 39}**