1. **漏洞代码定位过程：**

8b4: f90063ff str xzr, [sp, #192]，之后通过

8cc: 14000017 b 928 <main+0x90>跳转到

928: f94063e1 ldr x1, [sp, #192]

92c: d29cffe0 mov x0, #0xe7ff // #59391

930: f2a90ec0 movk x0, #0x4876, lsl #16

934: f2c002e0 movk x0, #0x17, lsl #32

938: eb00003f cmp x1, x0

这些是循环条件判断语句。

最开始的时候，x1存储的是0，x0存储的是0x174876e7ff，此后只要x1存储的数值 <= 0x174876e7ff(#99999999999)就会执行循环。

循环里是一个斐波那契数列。

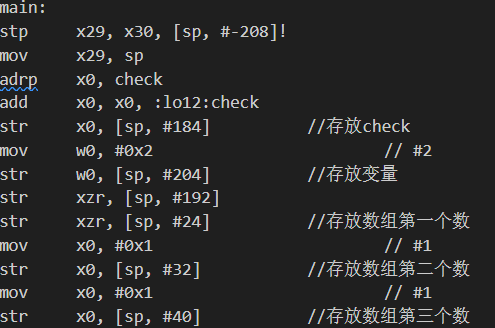
z =f[x- 1] + f[x - 2];

y = x++;

f[y] = z;

这里会出现数组的下标超过数组大小的情况

1. **缓冲区大小分析**

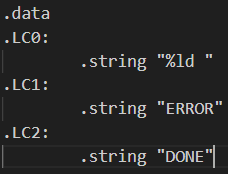


据图中注释可知，数组的起始地址是24，不能超过存放check的位置184。

并且从第一个数到第二个数的位置距离可知数组中的类型是long long型，占据8B

所以(184-24)/8 = 20，数组最多存储二十个数

1. **交叉引用列表**
2. 对于指向代码的交叉引用，通过跳转语句来看即可
3. 对于数据的交叉引用



把符号化成

把改成

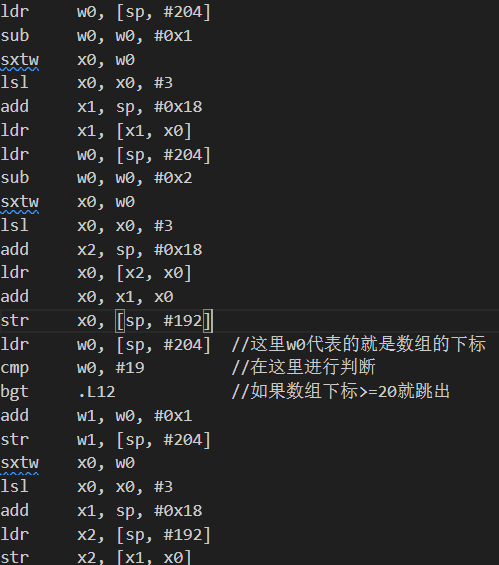
把改成

把改成

(3)对于函数调用，直接将其替换为对应的函数名即可

1. **漏洞修复过程**

修复漏洞就是让数组不溢出，所以我们需要在循环当中增加一个判断语句跳出循环



.L12:nop

输出结果：

