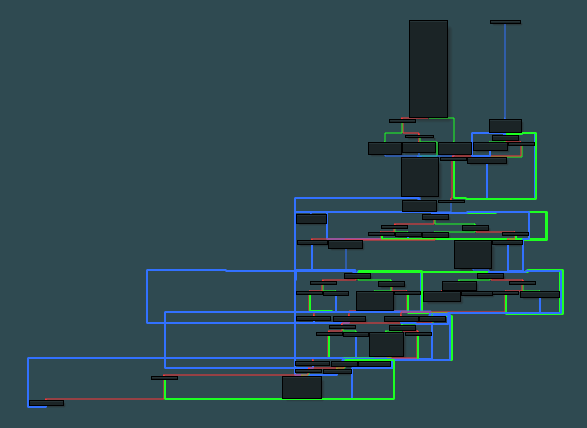
目标， 逆算法

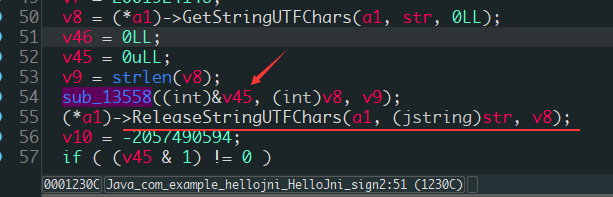
1. Java层逻辑比较简单，分别传15个和16个字符到sign2，直接去看native层



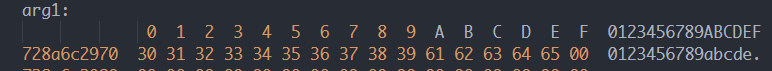
1. 混淆程度一般，直接从传入参数或者结果出发手撕

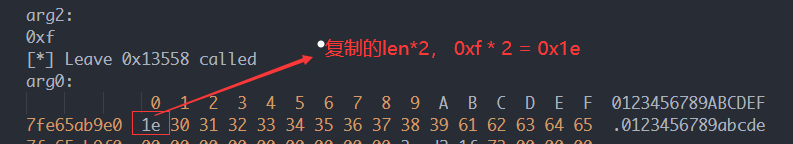


1. 这里把str先copy到v8然后执行sub\_13558之后又释放掉了，所以这个函数应该是对v8做了什么变换之后存到了v45



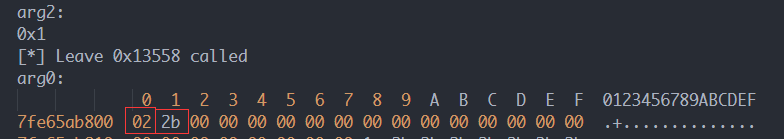
Hook一下这个函数，发现是把v8的长度\*2放在第一个byte上之后，把v8复制到后面

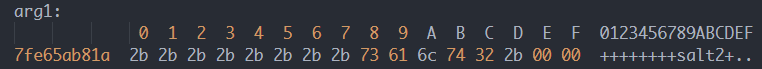


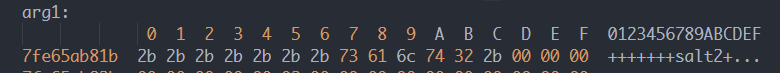


在HOOK这个函数的时候，发现在0x15cc8处也调用了这个函数，而且次数较多，比较有规律，每次把字符串取一个字符传入处理。

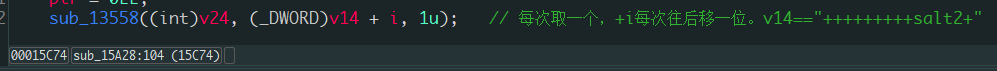






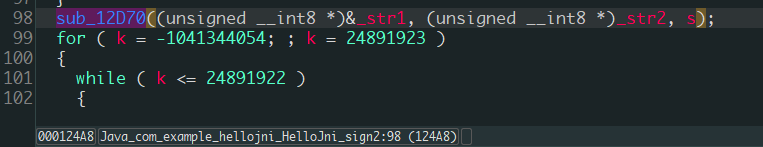


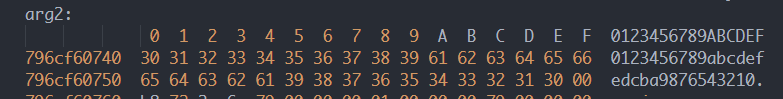
分别标记一下：



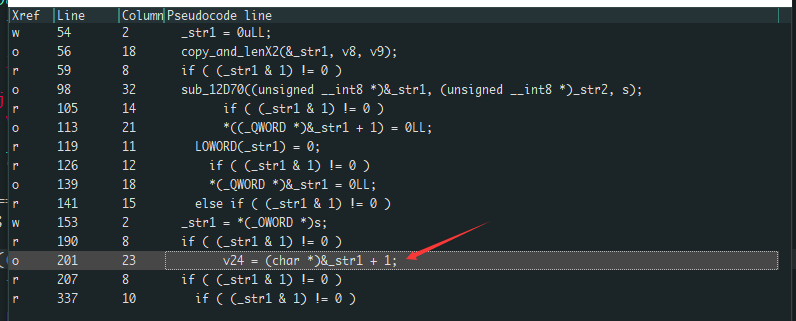


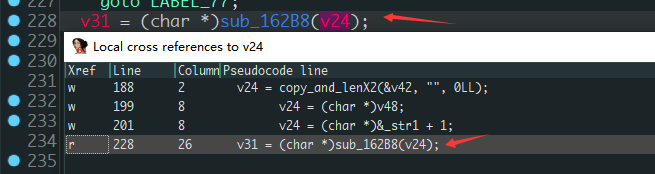
1. 接着hook sub\_12D70，因为同时与两个输入参数有关，发现这里参数个数有点问题，参数个数改成了3个，作用是将两个输入拼接起来后给第三个参数



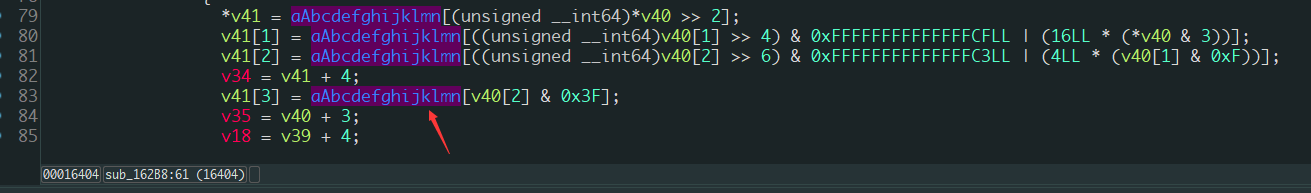


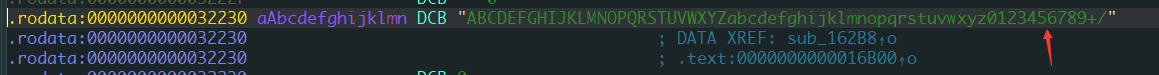
1. 下一处有点意思的是和str1有关，他将\_str1+1，刚好是输入赋给一个新的变量



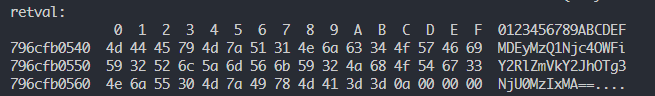


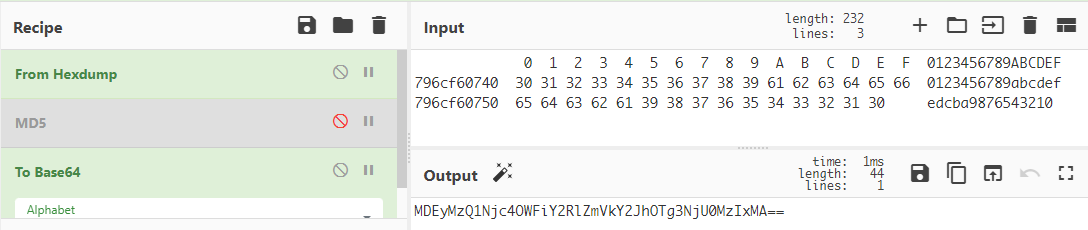
1. Sub\_162B8里有个表，看表估计是个base64



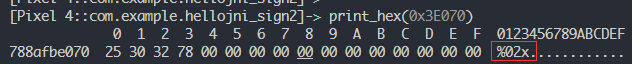


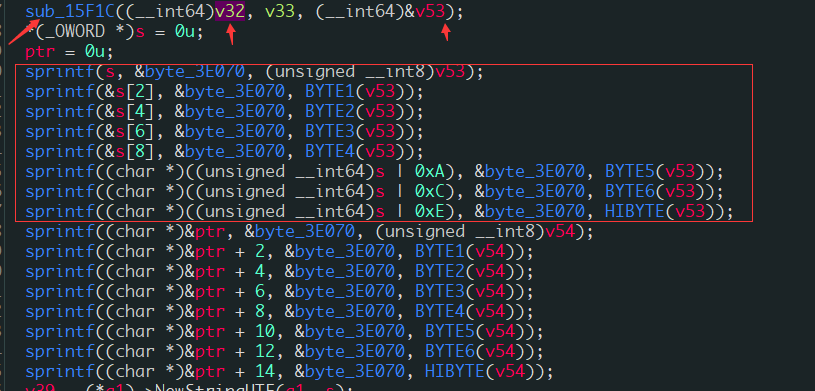
Hook之后，第一个参数是输入，第二个参数是输入长度，返回值是输入的base64编码





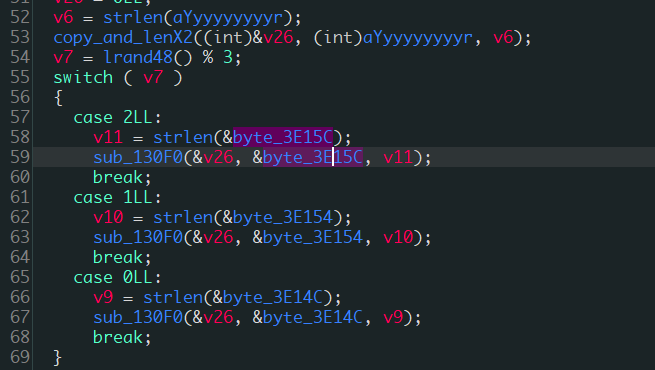
1. 继续跟踪输出的base64值，发现经过sub\_15F1C后，第三个参数与输出的字符串直接相关。

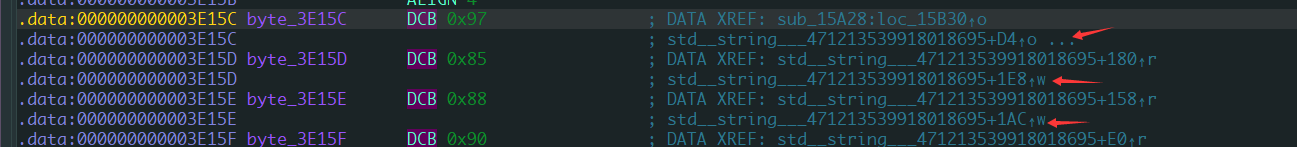


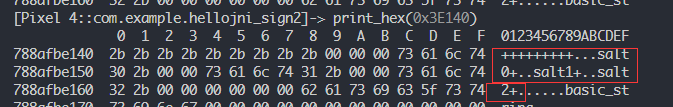


1. 进入sub\_15F1C

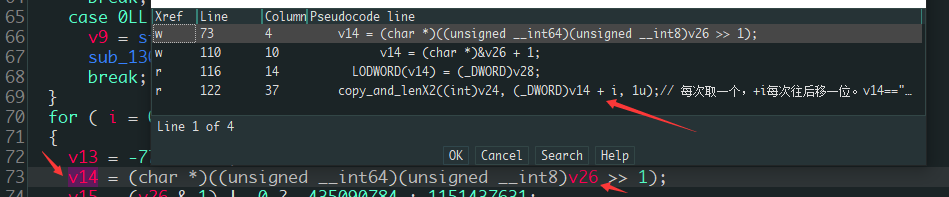
首先是发现有不少加密字符串



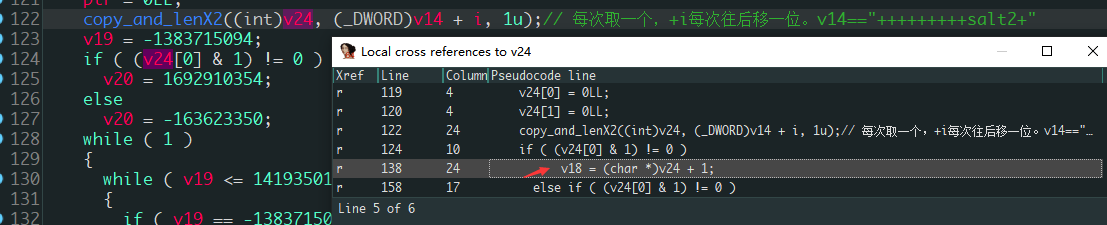


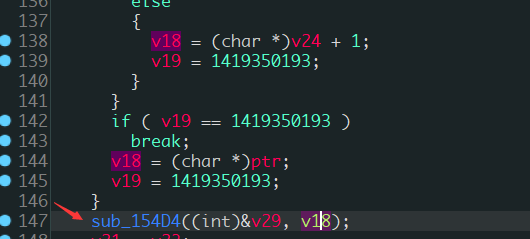


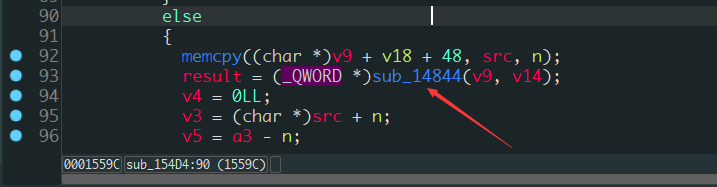
发现v26右移1位（这一位是代表长度）的值给了v14，而v14调用的地方有之前熟悉的标记



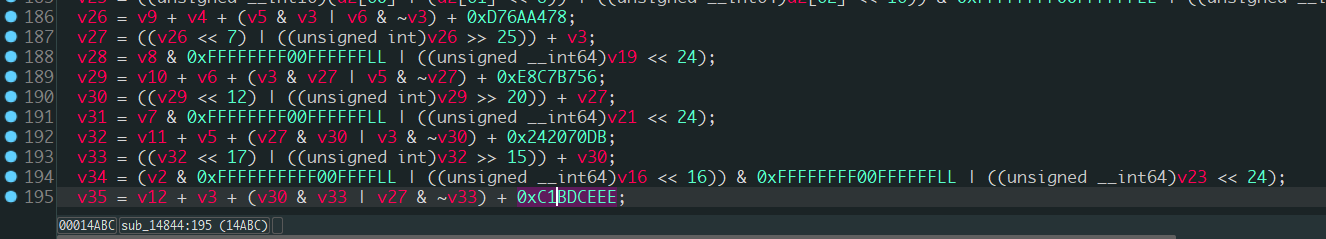
1. 继续通过交叉引用找



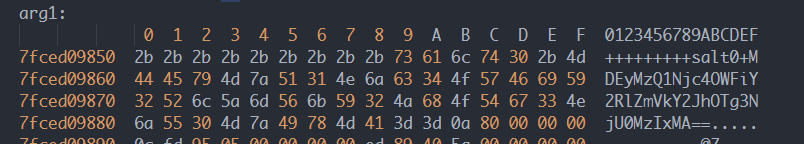


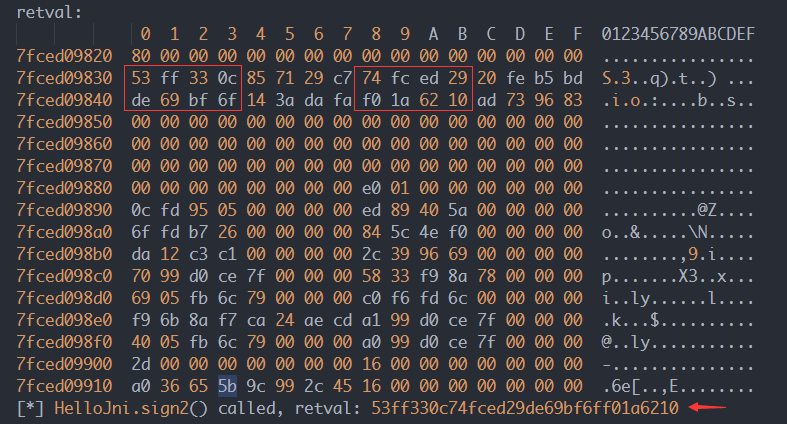


熟悉的MD5常量



1. Hook一下sub\_14844





验证一下：

