游戏发包流程

对于网络游戏来说，发送数据包是必不可少的一个环节，无论是走路还是喊话或者是其他功能都需要通过发包函数向服务器发送数据包。

以技能call为例，整个流程大致如下：

1. 接收到用户消息(键盘按下技能快捷键)
2. 调用技能call
3. 调用技能call2
4. 调用技能call3
5. 组装数据包
6. 对数据进行加密
7. send发包

技能call会有很多个，越往外层，功能越完整，但限制越多，越往里层限制越少，功能越强大。

例如在最外层的技能call1会检测是否到达冷却时间，当前人物状态等等，内层的技能call2直接释放技能，那么如果直接调用call2就可以达到无CD的效果。

所谓的变态功能只不过就是绕过游戏的一些限制，然后调用外层call达到目的。如果能找到最里层的call，或者直接发送封包，就可以直接绕过所有的限制。

借助这个特性我们可以在发包的函数下API断点，逆向分析出这个游戏的大部分功能。

当然也不排除底层对抗比较强的一些游戏，会重写三环的所有API，这个时候API断点就无效了。这种情况想要找到游戏的发包函数其实也很简单，我们下次再讨论

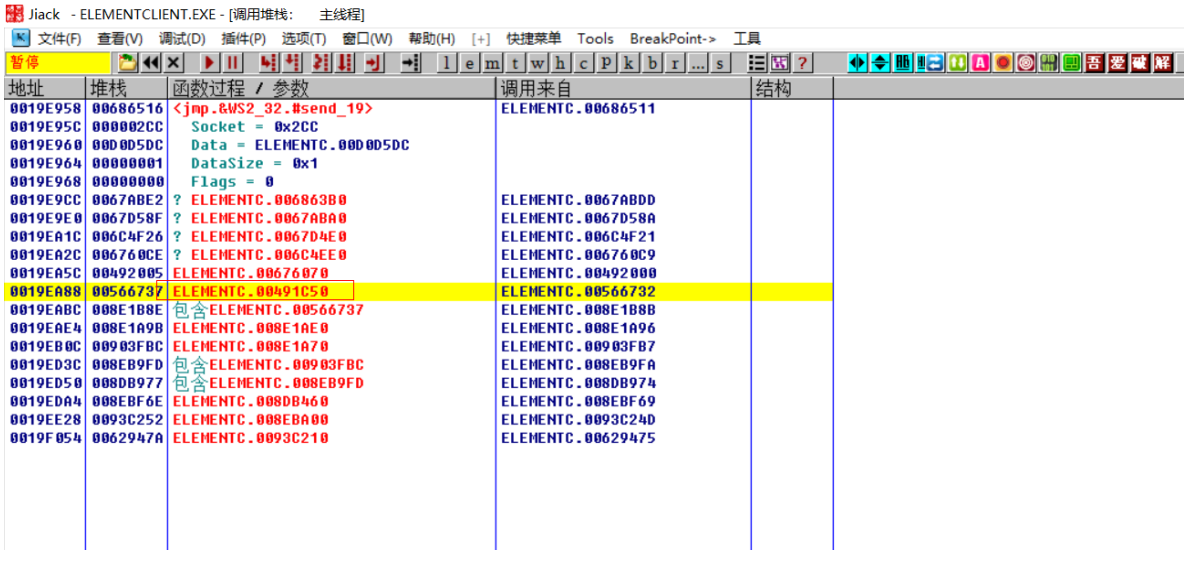
发包函数一共有下面的三个

```
send
sendto
WSASend
```

分析吃药call

这里以吃药为例，用OD附加游戏，并且下sendAPI断点

随便吃一个药品，这里需要记住以下药品的位置是在第22格



程序断下，打开调用堆栈，这里需要一个一个排除。最后可以确定吃药call是491C50，右键，显示调用

```
地址        HEX 数据        反汇编
00566718    E8 13220000     call ELEMENTC.00568930
0056671D    E9 98000000     jmp ELEMENTC.005667BA
00566722    8B5424 34       mov edx,dword ptr ss:[esp+0x34]
00566726    8BCB            mov ecx,ebx
00566728    52              push edx
00566729    6A 00           push 0x0
0056672B    E8 A045F9FF     call ELEMENTC.004FACD0
00566730    8BC8            mov ecx,eax
00566732    E8 19B5F2FF     call ELEMENTC.00491C50
00566737    84C0            test al,al
00566739    75 7F           jnz short ELEMENTC.005667BA
0056673B    83FF 01         cmp edi,0x1
0056673E    75 7A           jnz short ELEMENTC.005667BA
00566740    39BE AC090000   cmp dword ptr ds:[esi+0x9AC],edi
00566746    75 72           jnz short ELEMENTC.005667BA
00566748    8B8E B4090000   mov ecx,dword ptr ds:[esi+0x9B4]
```

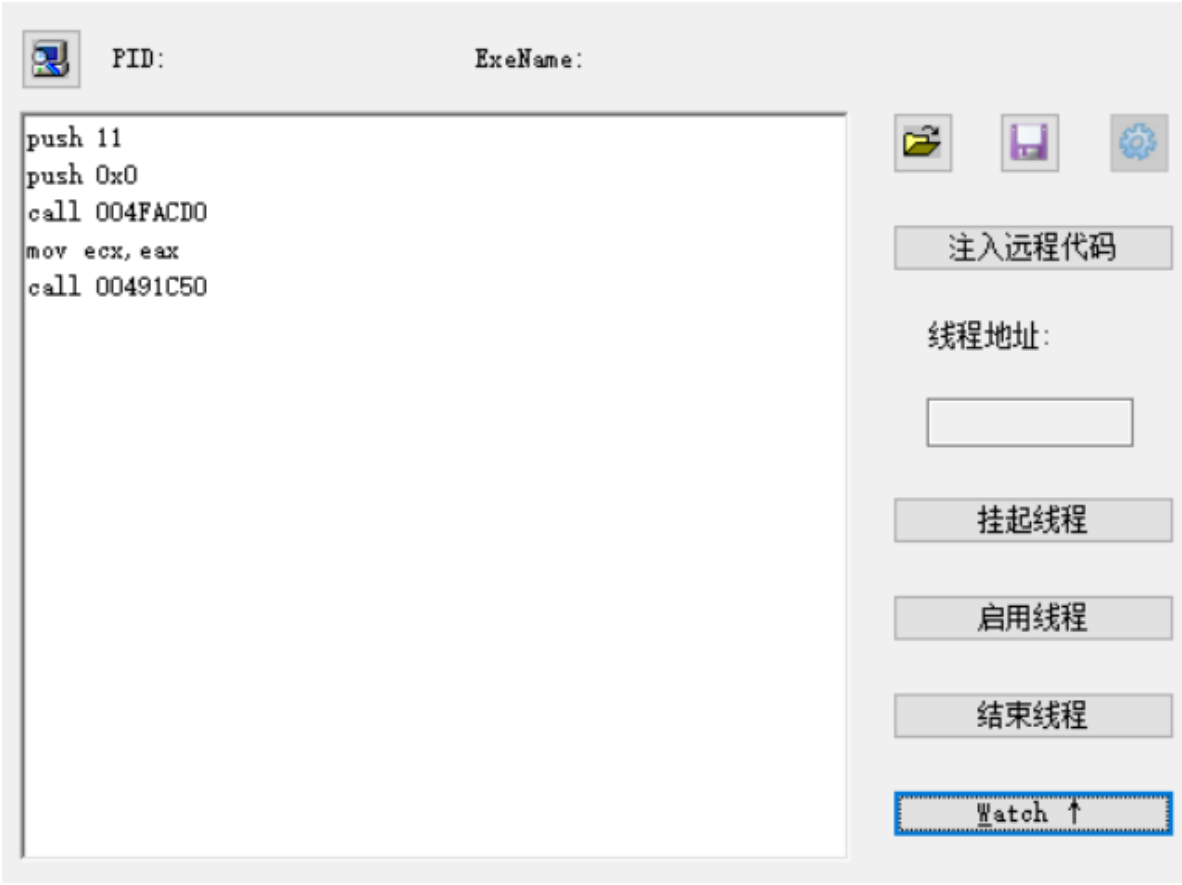
在这个地方下个断点 让程序断下，这个call很明显需要调用两个call。接下来分析代码，先进去上面的那个call看看

```
Jiack - ELEMENTCLIENT.EXE - [LCG - m主线程, 模块 - ELEMENTC]
文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools BreakPoint-> 工具
暂停
地址        HEX 数据    反汇编                                  注释
004FACD0    A1 1CDFD000 mov eax,dword ptr ds:[0xD0DF1C]          血量=[[[[0xD0DF1C]+0x1C]+0x28]+0x288]
004FACD5    8B48 1C     mov ecx,dword ptr ds:[eax+0x1C]         血量=[[[eax+0x1C]+0x28]+0x288]
004FACD8    8B41 28     mov eax,dword ptr ds:[ecx+0x28]         血量=[[ecx+0x28]+0x288]
004FACDB    C3          retn
004FACDC    90          nop
```

这个函数之前我们追血量基址的时候已经分析过了，那么大概可以猜到这个call就是获取当前人物的血量

然后再看edx的值是0x15，也就是十进制的21。药品的位置在第22格，如果从0开始计数的话，那么这个参数大概就是药品的位置了。

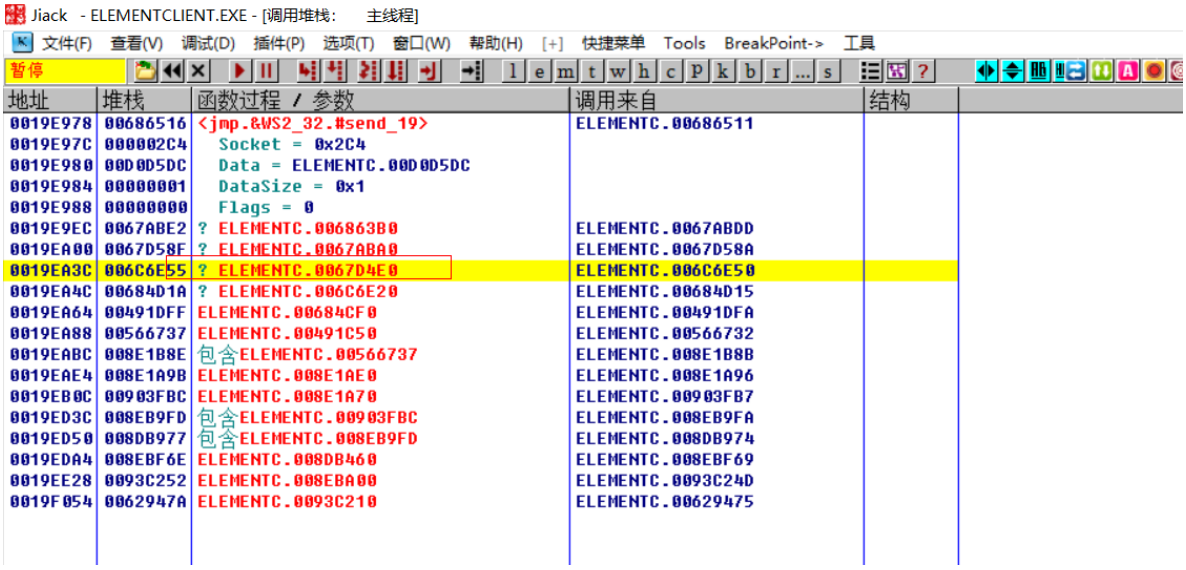
这个call的参数相对来说调用起来没有那么方便，如果有多个吃药call的可能性，那么在外层一定还有一个call，是可以通过药品的名称或者其他东西作为参数来调用的



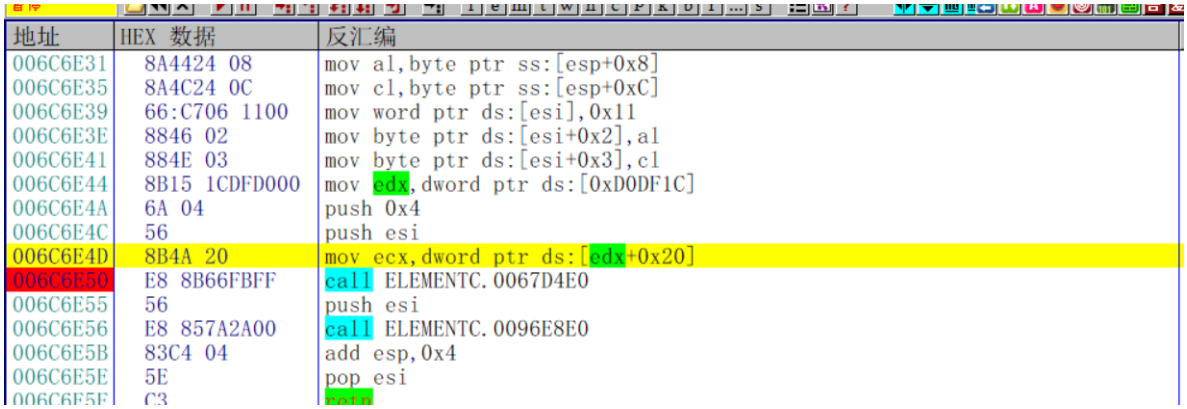
分析完成以后可以通过代码注入器来测试call是否有效

分析换装备call

同样的方法在send下断，然后更换装备让断点断下



点击调用堆栈，挨个排除，最后确定为67D4E0



接下来分析参数,ecx来自[edx+0x20]而edx=[0xD0DF1C]，这个数据不难跟

而esi的值是一个地址，指向的值多次测试有下面几种情况

```
02030011
03100011
000c0011
```

后面四位的0011是固定的。前面两位02 03 00指的是需要更换的部位，例如：上衣=02，鞋子=03，头盔01，武器=00

第三四位的含义是物品栏装备的位置，依旧是从0开始计数。代码就不写了，各位自行测试

总结

网络游戏的逆向分析的一个大的切入点就是发包函数，通过整个发包流程，可以逆向出绝大部分功能，这里只举了两个例子。

这个方法只适用一些简单的游戏功能，如果功能比较复杂，则会频繁向服务器发送数据包，通过这种简单的调用关系就不一定能找到关键call。

相关工具:

https://github.com/TonyChen56/GameReverseNote