

游戏发包流程
分析吃药call
分析换装备call
总结

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



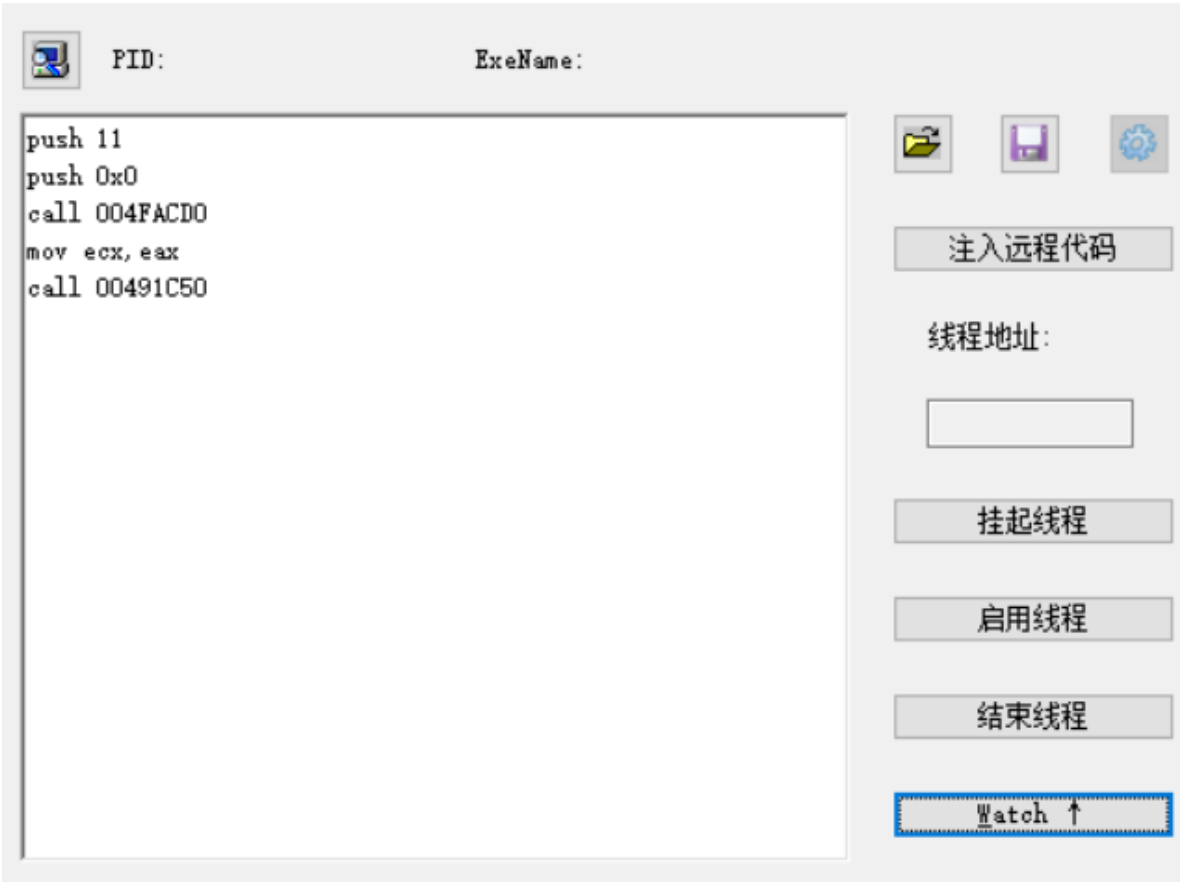
随便吃一个药品，这里需要记住以下药品的位置是在第22格

Jack - ELEMENTCLIENT.EXE - [调用堆栈: 主线程]

地址	堆栈	函数过程 / 参数	调用来自	结构
0019E958	00686516	<jmp.&WS2_32.#send_19>	ELEMENTC.00686511	
0019E95C	000002CC	Socket = 0x2CC		
0019E960	000005DC	Data = ELEMENTC.000005DC		
0019E964	00000001	DataSize = 0x1		
0019E968	00000000	Flags = 0		
0019E9CC	0067ABE2	? ELEMENTC.0067ABD0	ELEMENTC.0067ABD0	
0019E9E0	0067D58F	? ELEMENTC.0067ABA0	ELEMENTC.0067D58A	
0019EA1C	006C4F26	? ELEMENTC.0067D4E0	ELEMENTC.006C4F21	
0019EA2C	006760CE	? ELEMENTC.006C4EE0	ELEMENTC.006760C9	
0019EA5C	00492005	ELEMENTC.00676070	ELEMENTC.00492000	
0019EA88	00566737	ELEMENTC.00491C50	ELEMENTC.00566732	
0019EABC	008E1B8E	包含 ELEMENTC.00566737	ELEMENTC.008E1B8B	
0019EAE4	008E1A9B	ELEMENTC.008E1AE0	ELEMENTC.008E1A96	
0019EB0C	00903F8C	ELEMENTC.008E1A70	ELEMENTC.00903F87	
0019ED3C	008EB9FD	包含 ELEMENTC.00903F8C	ELEMENTC.008EB9FA	
0019ED50	008DB977	包含 ELEMENTC.008EB9FD	ELEMENTC.008DB974	
0019EDA4	008EBF6E	ELEMENTC.008DB460	ELEMENTC.008EBF69	
0019EE28	0093C252	ELEMENTC.008EBA00	ELEMENTC.0093C24D	
0019F054	0062947A	ELEMENTC.0093C210	ELEMENTC.00629475	

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

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

Jiack - ELEMENTCLIENT.EXE - [调用堆栈: 主线程]

地址	堆栈	函数过程 / 参数	调用来自	结构
0019E978	00686516	<jmp.&WS2_32.#send_19>	ELEMENTC.00686511	
0019E97C	000002C4	Socket = 0x2C4		
0019E980	00D0D5DC	Data = ELEMENTC.00D0D5DC		
0019E984	00000001	DataSize = 0x1		
0019E988	00000000	Flags = 0		
0019E9EC	0067ABE2	? ELEMENTC.006863B0	ELEMENTC.0067ABD0	
0019EA00	0067D58F	? ELEMENTC.0067AB80	ELEMENTC.0067D58A	
0019EA3C	006C6E55	? ELEMENTC.0067D4E0	ELEMENTC.006C6E50	
0019EA4C	00684D1A	? ELEMENTC.006C6E20	ELEMENTC.00684D15	
0019EA64	00491DFF	ELEMENTC.00684CF0	ELEMENTC.00491DFA	
0019EA88	00566737	ELEMENTC.00491C50	ELEMENTC.00566732	
0019EABC	008E188E	包含 ELEMENTC.00566737	ELEMENTC.008E188B	
0019EAE4	008E1A9B	ELEMENTC.008E1AE0	ELEMENTC.008E1A96	
0019EB0C	00903FBC	ELEMENTC.008E1A70	ELEMENTC.00903FB7	
0019ED3C	008EB9FD	包含 ELEMENTC.00903FBC	ELEMENTC.008EB9FA	
0019ED50	008DB977	包含 ELEMENTC.008EB9FD	ELEMENTC.008DB974	
0019EDA4	008EBF6E	ELEMENTC.008DB460	ELEMENTC.008EBF69	
0019EE28	0093C252	ELEMENTC.008EBA00	ELEMENTC.0093C24D	
0019F054	0062947A	ELEMENTC.0093C210	ELEMENTC.00629475	

点击调用堆栈，挨个排除，最后确定为67D4E0

地址	HEX 数据	反汇编
006C6E31	8A4424 08	mov al,byte ptr ss:[esp+0x8]
006C6E35	8A4C24 0C	mov cl,byte ptr ss:[esp+0xC]
006C6E39	66:C706 1100	mov word ptr ds:[esi],0x11
006C6E3E	8846 02	mov byte ptr ds:[esi+0x2],al
006C6E41	884E 03	mov byte ptr ds:[esi+0x3],cl
006C6E44	8B15 1CDFD000	mov edx,dword ptr ds:[0xD0DF1C]
006C6E4A	6A 04	push 0x4
006C6E4C	56	push esi
006C6E4D	8B4A 20	mov ecx,dword ptr ds:[edx+0x20]
006C6E50	E8 8B66FBFF	call ELEMENTC.0067D4E0
006C6E55	56	push esi
006C6E56	E8 857A2A00	call ELEMENTC.0096E8E0
006C6E5B	83C4 04	add esp,0x4
006C6E5E	5E	pop esi
006C6E5F	C3	ret

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>