CROSS†CHANNEL复刻版汉化全教程

**·MewCatcher**

## 复刻版与老版的对比

作为这次负责管理汉化CROSS†CHANNEL项目的PM，我也同时负责程序的破解。我们本来是准备汉化老版C†C的，但是在月初和“童话karas”商量开坑事宜的时候，有热心网友告诉我们复刻版在2012年11月22日放出（也就是月底）。如果开坑的话，是否应该以复刻版开坑呢？本人深知自己逆向技术很渣（P.S.老版的破解是由入门教程*So You Want to be a Hacker?* 教会的），所以想等复刻版出来之后试试，这期间我也恶补了很多知识。但是22号之后的几天，网上还没有放源，再加上已经有很多有热情的网友加入了我们的团队，开坑迫切，所以我们决定先以老版开坑。结果网上没几天就放源了，我第一时间下载下来研究了一下下，发现不是很难破解的，也发现有些小发现，于是决定作为团队成果发布的一部分——写出本系列破解教程，希望大家能喜欢。本人实在是小菜，**如有错误，也敬请大牛们指点**。

下面先把本系列教程的目录公布一下：

Part01 - C†C复刻版与老版的对比

Part02 - 系统语言验证、字符显示、字体修改相关

Part03 - 数据包结构分析、解包与封包

Part04 - 文件无法下断原因分析及解决方案

Part05 - 文本、PNG解密及加密

Part06 - 谈谈任务分配及业内默认规范，以及项目管理

Part07 - 数据的修改

Part08 - 存档分析及进度统计方法

Part09 - \*双语切换标签的制作（可能很麻烦，涉及到Hook）

Part10 - 补丁安装包的制作

由于本游戏尚未完全破解，所以以上内容将进行不定期更新，以丰富广大玩家的“漫长”等待生活O(∩\_∩)O~ 扩大些知识面吧。因为我们没有英化版人员那样的高效（他们3个月完成整个坑）。

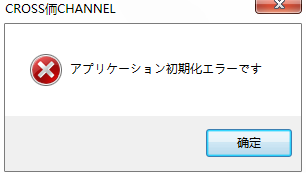
好了，言归正传。大家应该都知道了，这次复刻版是WillPlus出品的，原来是FlyingShine，引擎自然会变的。30号终于看到源了，下下来研究一下，首先看到的是图标变得华丽了，用AppLocal载入安装之后，看到文件目录是“CROSS†CHANNEL”，老版的是“cc”。当然画面由原来的640x360增加到了800x600，而且有明显的重绘迹象，动画效果也增加了很多，还是能跟得上时代的。



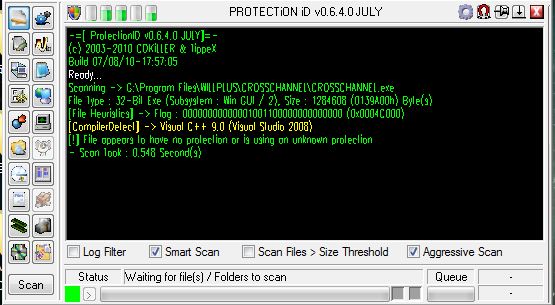
看看封包，之前有接触过WillPlus旗下Propeller的一些作品，看到的都是mpk格式的封包，这次是arc（英文是archive，所以光凭此扩展名看不出什么，例如Narcissus3rd的封包也是arc，但是结构却很不相同），老版的是pd格式的，所以断定是新格式，心一沉，本菜菜感到很有压力。抓紧研究了的说。

## 二、系统语言验证、字符显示、字体修改相关

话说，拿到游戏首先当然是要运行喽，我的操作系统是Win7-32bit简体中文版本的，双击快捷方式，弹出如下坑爹对话框：

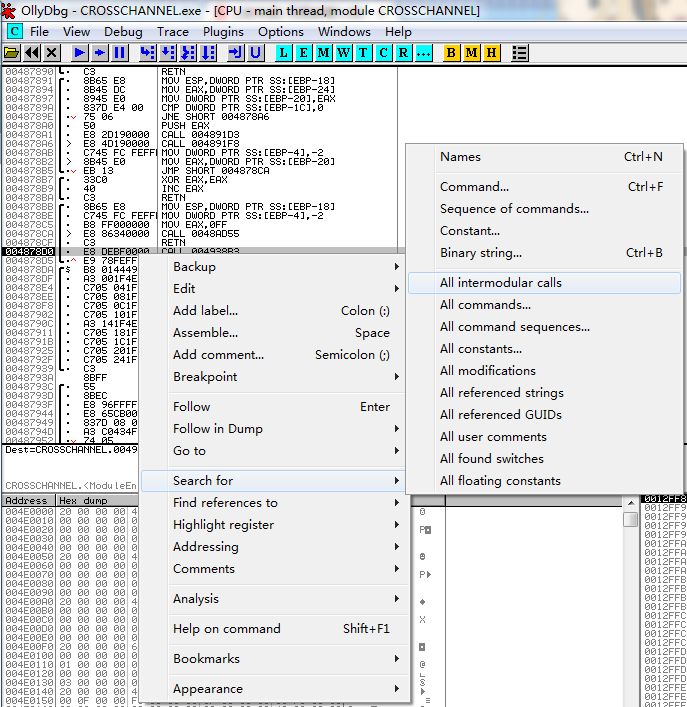


老版游戏不插光盘会弹出对话框，对于日语菜菜我来说，这有多种可能性，所以我先插虚拟光盘试试，问题依旧，真是的，估计是要修改区域设置了，从来没做过啊。不过，这次不用修改为免CD了，哈哈。既然猜到是要修改区域设置，那首先用AppLocal载入试试，成功进入游戏，这样就确定是由于区域验证导致的了。当然，这是我第一次修改系统语言设置，所以我不知道要在哪下断，所以按照基本流程，我们先用个东西查壳，这里我用PEid没查到，但是用Pid查出来是这样的：

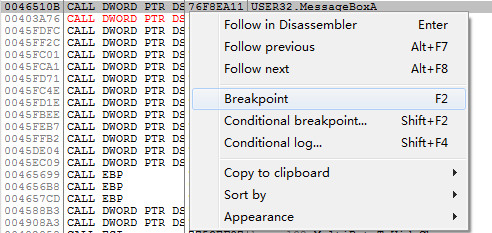


所以断定没有加壳，于是我们OD载入，在弹出那个“初始化失败”对话框的地方下个断吧。

首先用OD反汇编（我这里用的是OD201d）：

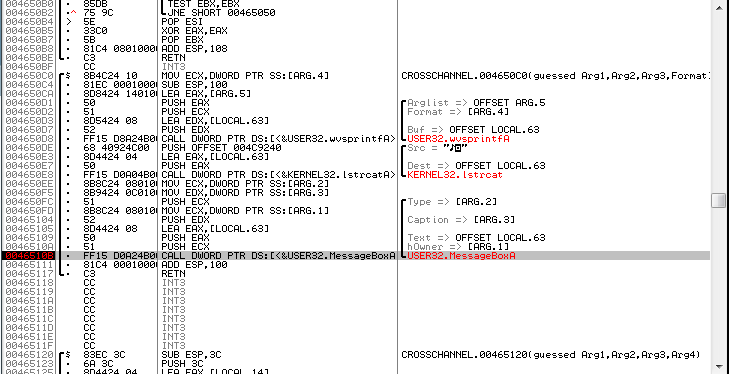


右击反汇编窗口，选择【查找】🡪【所有模块间调用】，进入如下界面：

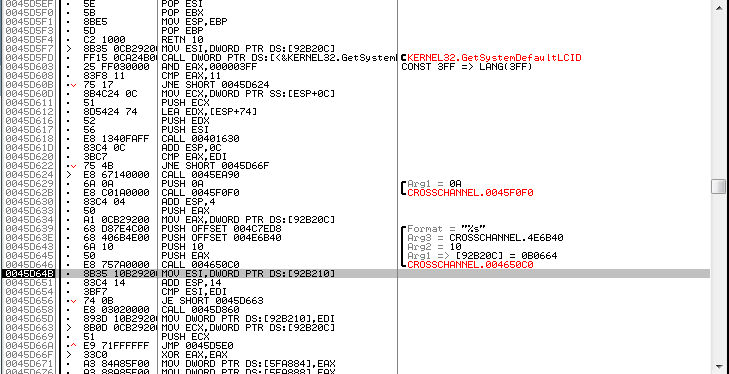


找到MessageBoxA，右击，下断，然后点运行就行了。（注意，这里这样说是不严谨的，实际上我还调查了游戏目录下的CROSSCHANNEL.dll和GameuxInstallHelper.dll的引用函数，其中可能性不太大，所以这里没有阐述。）

过掉界面设置之后，会断在0x0046510B这个地方：



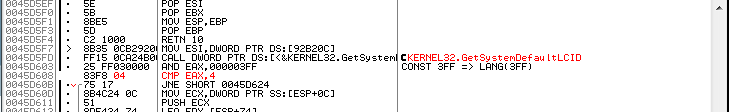
上下翻翻没什么有用信息，那么我们继续走，按F10单步走，走到RETN，返回到之前调用的地方，如下图所示：



这时候已经弹出“初始化错误”的对话框了，但是往上翻翻，能看到一个非常可疑的API调用，那就是在0x0045D5FD的GetSystemDefaultLCID，查询MSDN发现就是这家伙惹的祸，它是获取操作系统语言的API，其返回值就是系统语言代号，其中英文是0x409，日文是0x411，简体中文是0x804，繁体中文是0x404。如图可以看到，0x0045D603的地方，该程序将语言代号返回值和0x3FF做了与运算，并判断得到结果是否为0x11。

那么我们可以有很多种改法，下面提供两种：

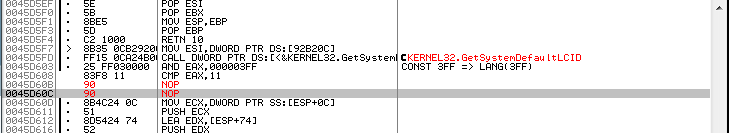
1.让简体中文的代号0x804跟0x3FF做与运算，并将结果改到判断中，如图所示：



但是这种方法只能使其支持简体中文操作系统，换成繁体的就不行了，有很大局限性。

那么我们再看看第二种方法：

2.让JNE失效，即“即使系统不是日语，也不跳转”，那么用空白nop填充即可，如图：



这样修改，系统就能支持所有语言的操作系统了。

但是！！！尼玛还真是坑爹无止境啊！！！欺负我小菜菜啊！！！

又是这个家伙：



怎么可能能能能能能能？？？理论上完全正确啊！！！！

经过我三番五次的尝试，终于发现问题出现在游戏文件夹上了，如图：

G:\_MewCatcher's Projects\汉化项目\CrossChannel复刻版\破文\2-9.png

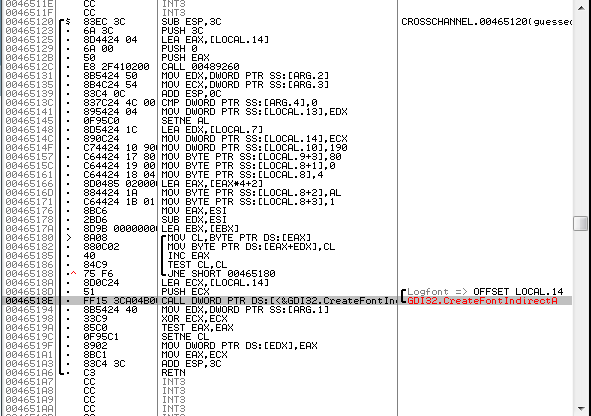
有一个不属于GB2312的字符，就是那个十字架号。没办法，只能改了它。

只改文件夹？没用！！！所谓无止境，我们得从注册表中根除它，初学者想找到相关的键值，就把几个根都翻一遍比较方便，或者直接搜索。我这里就直接把键值告诉你们：[HKEY\_CURRENT\_USER\Software\WillPlus\CROSSCHANNEL]，这里有个键叫InstallDir，先把游戏安装文件夹名称改掉，再在这里改一下游戏路径就从根本上解决问题了。（P.S.打开注册表的方法：【WinKey】+【R】输入RegEdit就行了，不区分大小写。）打开游戏，不错啊：



至此，系统语言验证就修改完成了！~

接下来，不知你们有没有发现现在能正常显示日文字体，其实这很有可能是创建文字句柄的时候设定区域值为日文地区的值，那么我么来看一下该程序有没有调用CreateFontIndirectA，还真有，那么就基本上可以确定没有使用宽字符了（P.S.宽字符就是一种能包含多种语言的字符统一编码，如：Unicode、UTF-8、UTF-16）。我们双击“模块调用界面”的该行，自动跳到调用的汇编代码处，如图：

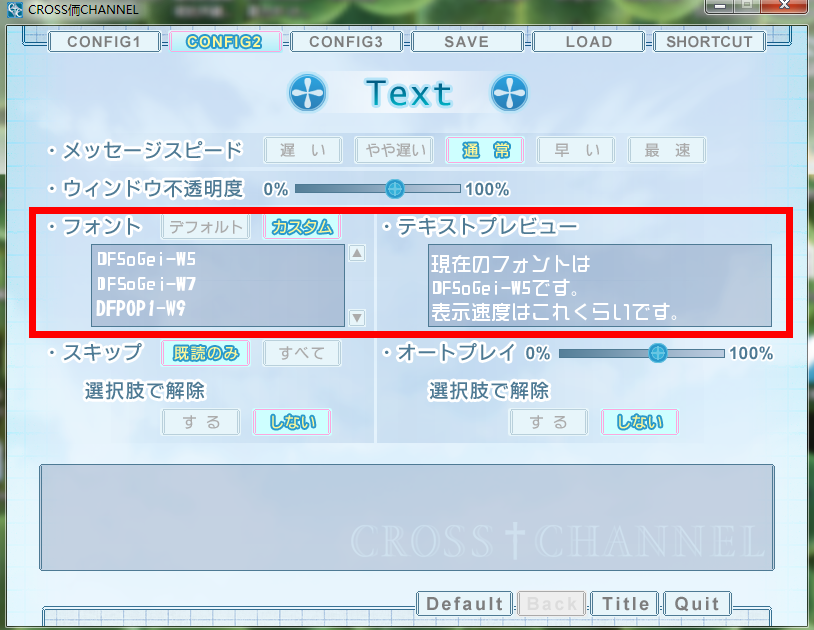


其实，很简单，你只要记住中文字符编码是0x86，日文是0x80就行了，朝上面找80这个字段，就很容易发现在0x00465157这个地址处有，那么我们就将它改成86就行了。其实对于一个陌生的API，最好查一下它的定义，然后按照C++的调用规定，从右向左依次传值，数数就能定位到了。

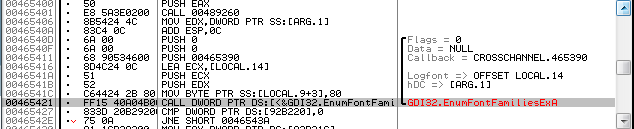
于是，保存之后，我们就将文字显示的问题解决了，看效果图吧：



好了，现在我们想尝试一下修改字体，结果发现都是日文字体：

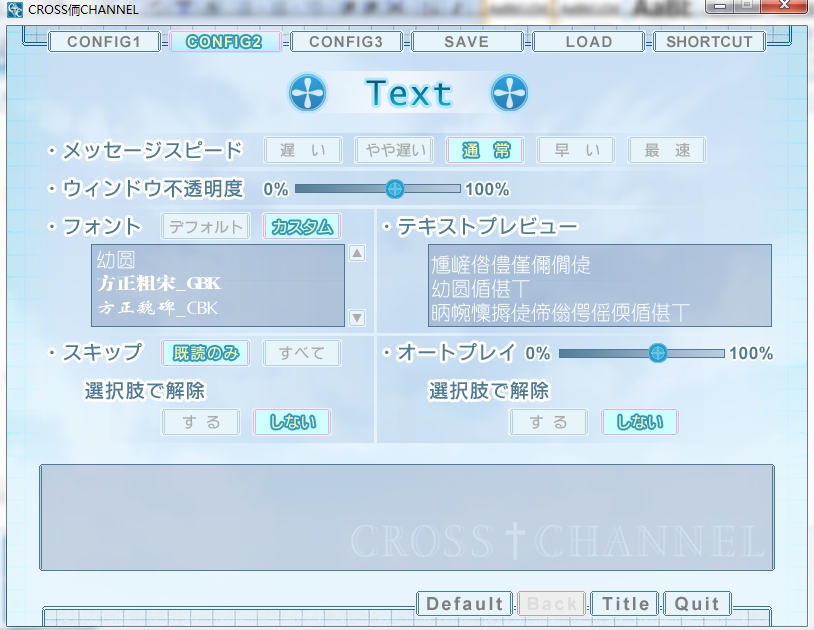


于是，到百度上搜一下“获取字体列表”，发现了几个关键词，其中最重要的是enum，于是在OD里面搜e，就发现这个东东了：EnumFontFamiliesExA，可疑函数，我们再次双击，来到调用处：



在0x0046541C地址处，又出现了80，哎哟喂，刚学的不是吗？试试改成86会怎么样？

哈哈哈哈哈哈哈……成功了：

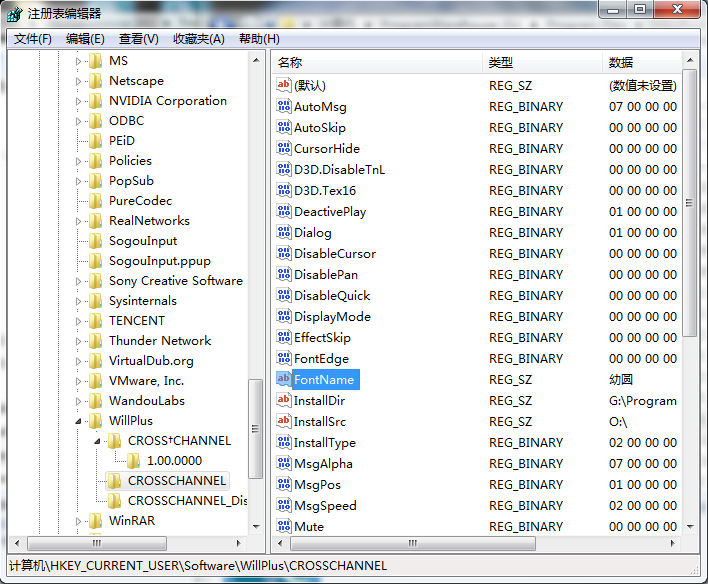


其实，这里不是巧合，跟字体相关的API，如果要用到区域的话，基本上都是一样的，日文0x80，中文0x86。

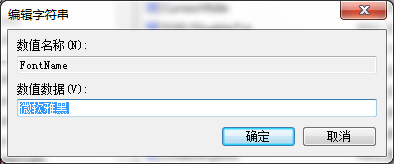
于是，字体的支持也就改成功了。

P.S.也许你会问，为什么没有“微软雅黑”啊？恕我直言，我也不知道……以后会补充的，后面有猜想。

不过，你如果对“微软雅黑”或者其他的字体有大爱，我给你们介绍种替代的方法，那就是看游戏把字体设置信息保存在哪了。首先浏览游戏所在目录，没有其他文件；再看我的文档有没有相关文件，也没有；那么我们就看注册表了。找到如图位置（其实和上面文件安装目录的地址是一样的）：



哈哈，有看到了有效键值，双击，把这个键值改成微软雅黑：



然后打开游戏看看，Perfect！



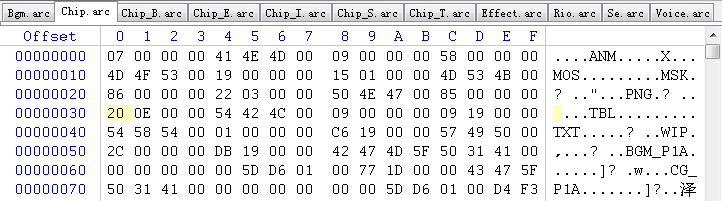
好了，现在总结一下，我觉得“微软雅黑没有出现的原因”是，“它不是系统等宽字体”，但是为什么不能显示非系统等宽字符呢？╮(╯▽╰)╭No idea…

## 三、数据包结构分析、解包、封包

通过调试手段解包一直都是很让人觉得性价比不高的方法，因为封包中的一些数据在游戏的局部运行中并不需要，这可能会让你纠结许久；而且，大多数情况下，解包出来的文件有的仍然是密文。所以，我们拿到一个封包，一般首先通过找规律归纳出封包结构。

不过，在中国，嘿嘿，你可以先试试用Crass提取，它能提取几百种游戏封包。不过，本游戏的它不能提取。

先看看目录里有多少.arc的文件，11个，相当给力，肯定足够分析了！于是我们用WinHex载入这些文件。这里先说明一下，WinHex是十六进制编辑器，他所能编辑的是文件的本质，也即二进制状态，甚至快捷方式也能编辑。放图了：

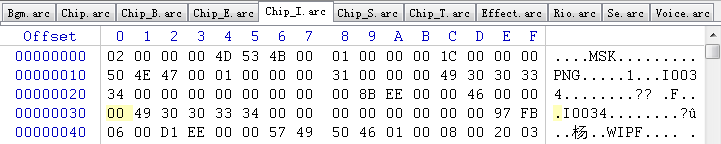


一般来说，一个封包包括如下部分：文件头标记、包内文件索引列表（文件头或尾）、包内文件实体。

于是，我就是受这个“一般经验”的影响，将文件头0x07000000当成文件头标记了，绕了很大的弯路，菜菜我在此不再献丑，直接说正确的归纳了。

我们拿个简单的封包来讲解一下归纳的结果：

1. 包内文件格式类型数目（4字节）
2. 包内文件格式类型详细列表（n \* 12字节 = 4字节格式名 + 4字节该格式文件数 + 4字节文件索引起始地址）
3. 文件名详细列表（n \* 21字节 = 13字节纯文件名 + 4字节文件大小 + 4字节文件偏移地址）
4. 文件实体部分



对应此图中的实现：

1. 0x02000000 = 2 个文件
2. 0x4D534B00 = “MSK” 文件类型名， 0x01000000 = 1 个文件， 0x1C000000 = 0x1C 为偏移地址；

0x504E4700 = “PNG” 文件类型名， 0x01000000 = 1 个文件， 0x31000000 = 0x31 为偏移地址。

1. 0x49303033340000000000000000 = “I0034” 文件名， 0x8BEE0000 = 35822 字节文件大小， 0x46000000 = 0x46 文件偏移地址；（属于MSK文件类型的）

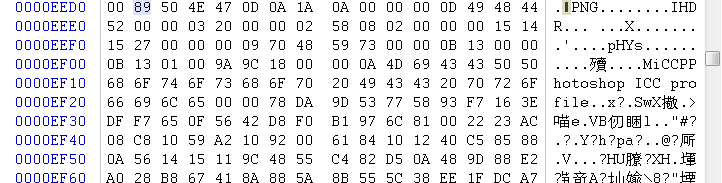
0x49303033340000000000000000 = “I0034” 文件名， 0x97FB0600 = 457623 字节文件大小， 0xD1EE0000 = 0xEED1 文件偏移地址；（属于PNG文件类型的）

1. 从0x46开始，是第一个文件I0034.MSK，以此类推……

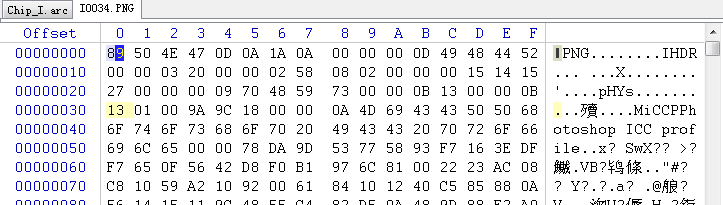
现在我要解释一下为什么0xD1EE0000 = 0xEED1，这个是Little-Endian的编码方式。与之对应的是Big-Endian的编码方式0xD1EE0000 = 0xD1EE0000。在此处，完全可以通过合理性判断得出是Little-Endian，因为0xEED1(61137)比0xD1EE0000(3522035712)更有可能是文件偏移地址。

所谓Little-Endian就是对于十六进制文本，左边是低位，右边是高位；Big-Endian则是左边是高位，右边是低位。比如说，对于十进制数123456789，最高位是1（亿位），最低位是9（个位）。

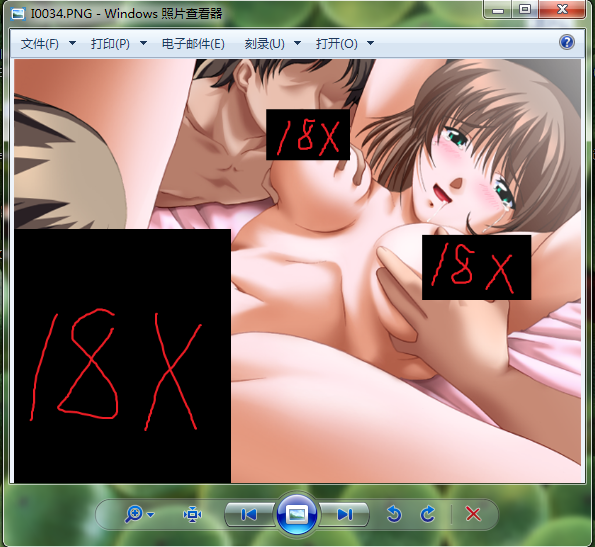
接下来，我来教你们怎么验证以上猜想是正确的。对于I0034.PNG文件，我们先定位到文件起始地址：



看到这个文件的文件头有”PNG”标识，基本上确定一半了，接着按照之前找到的规律，选取相应文件大小457623字节的块，一直到 0x7EA68 – 1 = 0x7EA67，复制到新的空文件中，保存为I0034.PNG文件：

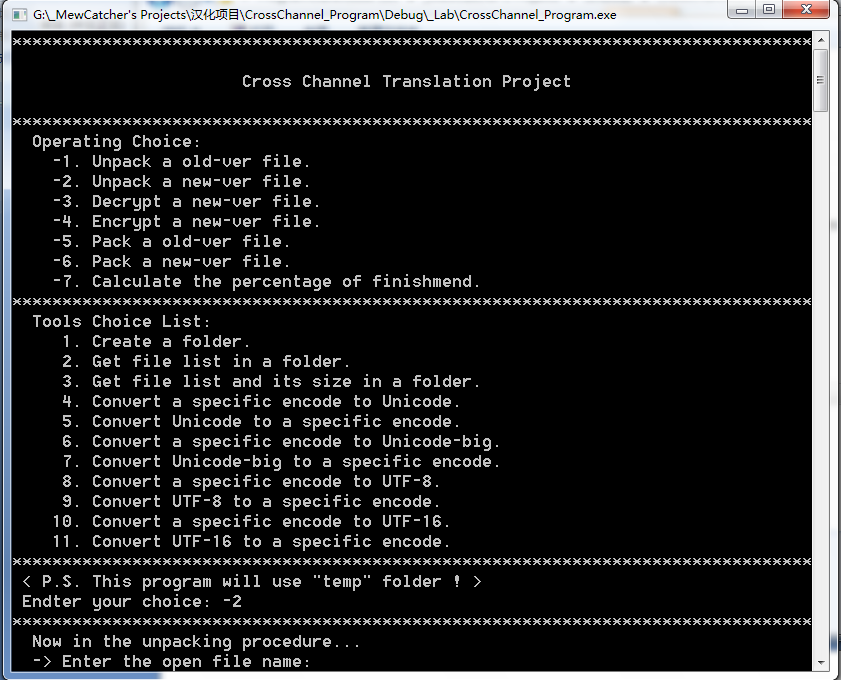


OK，打开文件看看。。。。。。额。。。。。。



出乎意料的图啊。。。。。。不过，总之是成功的。之后就是写出个解包的程序了。

解包程序源码就不放了，因为封装在类里面，不太好贴出来，而且写的确实挺烂。这里放出个截图（英语用错太多，懒得修改了，勿笑）：



提取结束后像这样：

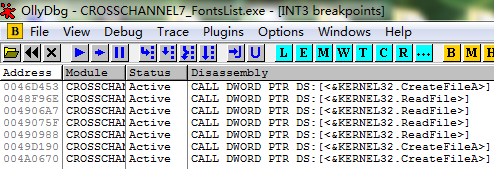


封包就是解包的逆向过程，这里不赘述了（节约时间。。。），不过要注意，要将相同扩展名的聚集在一起封包。

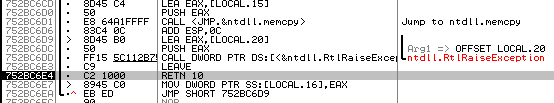
## 四、文件下断失败原因分析及解决方案

前面的文章多少有些暗示本文，确实，准备调试加密算法的时候却发现文件没法下断，无计可施，期末考试又快到了，只能请wiki908队友通过密码学知识先行分析一下，成果放在下一篇。之前在澄空上也发了咨询帖，感谢热情的“损疾”前辈指教，原来只是一个关于“异常”的小问题，只要在提示异常的时候按Shift+F7/F8/F9继续调试或者将最近的异常加入忽略列表就行了。不过在咨询前我已经一步一步找到了大概原因，下面就分享一下我是如何找到的错误原因的吧。

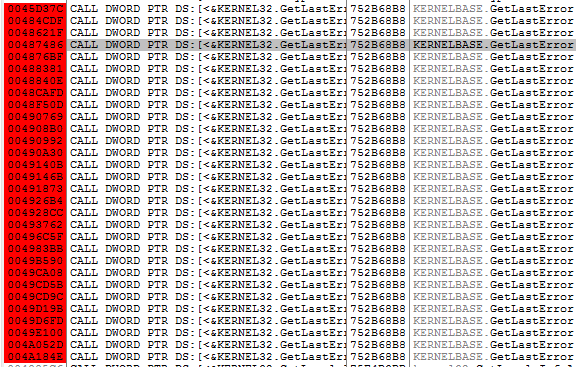
我们调试解密算法首先肯定要在文件相关的API下断点，然后单步走到加密处，于是我就下了如下图的几个断点：



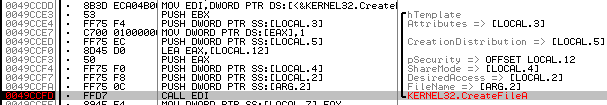
断下来之后，我就F8和F7单步跟踪，结果跟到如下图所示的情况——RaiseException收集异常：



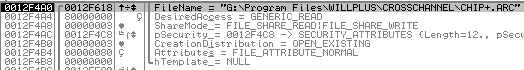
接着运行就是Win7常见的“程序已经停止运行”，不放图了，很别扭的界面，想了半天，在API列表里面翻弄着，突然看到了GetLastError，觉得可以试试跟踪这个函数，因为这个函数的参数在OD里面是能看到宏定义的值的，于是在所有的GetLastError上都下断点了，如下图：



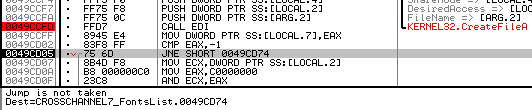
首先让程序停在第一个CreateFileA，如下图：



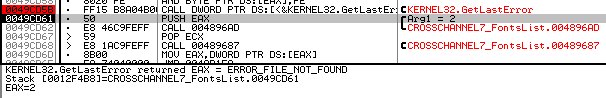
然后我们单步跟踪，看到右下角堆栈窗口显示打开的是CHIP+.ARC，如下图。感到很奇怪啊，游戏文件夹里只有CHIP.ARC，怎么多了一个加号呢，本来没有引起注意，因为很多面向对象的语言字符串都是很执行加法连接操作的，所以就不敏感了。



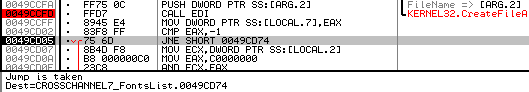
于是单步走，紧跟着的一个跳转没有实现，如下图：



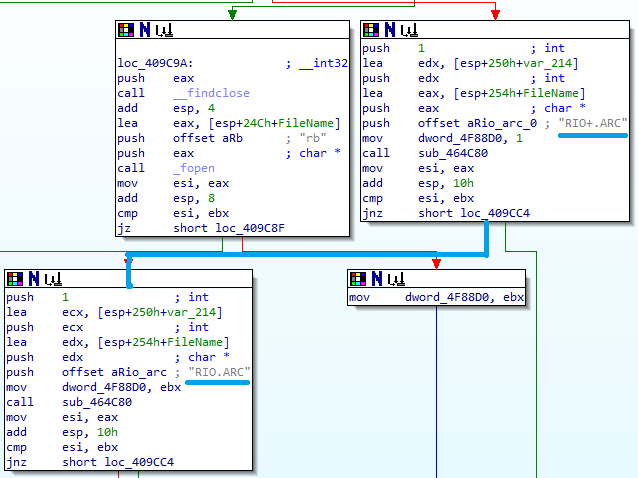
然后跳到了老远处，是GetLastError，往下走一点点，看到EAX=ERROR\_FILE\_NOT\_FOUND，如下图：



于是我就觉得前面那个CHIP+.ARC有蹊跷，索性把游戏文件夹下的CHIP.ARC文件名改成了CHIP+.ARC，于是就成功了，CreateFileA下面的跳转就实现了，如下图：



这当然是有原因的，游戏首先判断文件夹下是否有CHIP+.ARC，没有的话就找CHIP.ARC，但是没有成功打开文件夹会记录异常的，所以在OD里面没有选择忽略异常系统就会让程序崩溃， 用IDA看一下游戏判断文件的顺序：



RIO+.ARC没有找到之后JNZ是跳到读取RIO.ARC的程序段，个人认为RIO+.ARC是程序员调试用的接口，不过这个问题总算是安全解决了。

小提示：汉化包是否可以就用添加+的方式放在游戏目录呢？（答案是不建议，这样日文游戏就没法运行了，拖出去……）

## 五、文本、PNG解密及加密

我们团队成员wiki908在第一时间通过密码学手段将文本的加密方式分析出来了，也写了一个破文，但是由于文中有很多日文，网易没法正常显示（比如你看我们的C†C中间的加号就没法显示），百度文库又说是侵权作品。所以就不单独出日志了，这里发布一个下载链接：<http://note.youdao.com/share/?id=4066cfb9920b3c8055b75519470e8688&type=note>

下面这篇文章，我将从反汇编逆向的角度来解析封包的加密情况。

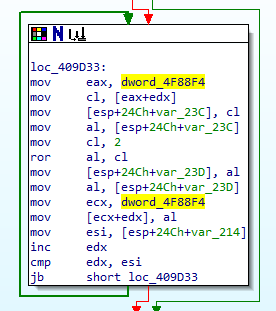
先来看看文本的加密情况。这次文本加密很神奇，与wiki908的分析结果不完全相同，也弥补了我们知识和经验上的一些不足，大家听我道来。

已经解决过了程序无法下断的问题，那么就可以尽情下断点了，我仍然是现在CreateFileA上下断，然后发现打开的文件都是图片之类的文件包，我索性撤除所有断点，先让程序进入主界面，然后在点击“开始游戏”之前把CreateFileA断点下上，点击按钮，成功断下了RIO.ARC，从堆栈窗口可以看到打开的文件是哪一个，发现打开了很多乱七八糟的文件，我跳过了很多ReadFileA，最终我选择了SCR.WSC这个文件，因为这个文件肯定是Script的缩写（拖出去……）……因为这个文件会指引程序去找哪个文件，所以这个文件必须解密，前面打开的乱七八糟的文件都不一定解密了。在ReadFileA断下来之后，单步走。

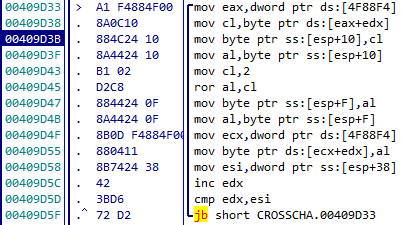
这里我想提示一点，想我这样调试菜菜（IDA用不好也看不懂，OD定位不好等各种菜），建议调试的时候同时开IDA和OD。当OD进入了不知道是不是应该看的部分（反汇编程序里有很多是没必要看的），你就可以通过查找OD地址在IDA的存在形态，判断是否有阅读价值。

例如：我现在运行到0x00409999了，我不知道这里是什么乱七八糟的东西，我就可以看看IDA里面这个地址显示的是什么东西，如果发现这个地址对应的是引用的外部函数，或者是灰色等不正常颜色，你就知道这里是八卦阵，来了会晕死掉的，可以在OD里按Ctrl+F9运行到这个调用的返回。

于是我就是这样，来到了一个循环（我们知道，处理文本肯定是循环，而且是小循环），在觉得像好东西的时候，我又通过查IDA的方式，确定了此部分确实是解密部分，下图是IDA环境下看到的解密部分：



下图是OD环境下看到的解密部分：



其实我看不懂的，通过查书，和OD反汇编窗口底部的实时状态信息，我猜到一些，下面把分析结果贴出来给大家了解了解：

00409D33 |>/mov eax,dword ptr ds:[4F88F4] ; EAX存的是DS:[4F88F4h]的首字节位置，相当于字符串指针指向

00409D38 |.|mov cl,byte ptr ds:[eax+edx] ; 取一个字节送到CL中，访问用EAX基址 + EDX偏址

00409D3B |.|mov byte ptr ss:[esp+10],cl ; 复制未解密字节到VAR1(SS:[ESP+10h])

00409D3F |.|mov al,byte ptr ss:[esp+10] ; 将未解密字节送到AL中

00409D43 |.|mov cl,2 ; 位移数赋值到CL中

00409D45 |.|ror al,cl ; 执行右位移2位，AL处理完毕

00409D47 |.|mov byte ptr ss:[esp+F],al ; 将解密后的字节送到VAR2(SS:[ESP+0Fh])

00409D4B |.|mov al,byte ptr ss:[esp+F] ; 循环复制一下，AL的值仍然是解密后的

00409D4F |.|mov ecx,dword ptr ds:[4F88F4] ; 从原文中取4字节放到ECX中，DS:[4F88F4h]放的是文件内容

00409D55 |.|mov byte ptr ds:[ecx+edx],al ; DS:[ECX+EXX]是内容，将解密后的内容替换到原文件内容中

00409D58 |.|mov esi,dword ptr ss:[esp+38] ; 将文件长度赋值到ESI中

00409D5C |.|inc edx ; 循环索引变量（也是数组索引变量）增加1

00409D5D |.|cmp edx,esi ; 判断循环索引变量与文件长度大小的关系

00409D5F |.\jb short CROSSCHA.00409D33 ; EDX<ESI时转移

这里我们调试发现解密指令只有一个ROR AL, CL; 并不像wiki908分析的一样（如果你看过的话），先加一些0xFF，然后右位移2位。为什么呢？我用windows自带的计算器尝试了一下（它由Ror按钮），发现了很神奇的是这个右位移是以8位为单位，尾数滚到首部填充，而我们的C、C++语言中的移位运算符是填充0的。

于是我通过查书，了解到：

1. ROL/ROR是不含进位标志位CF的循环左/右移位，移位是8位就这八位滚动，例如：01000010b Ror 2 = 10010000b。
2. RCL/RCR是含进位标志位CF的循环左/右移位，可以将CF的内容移入，所以可以利用它实现多字节的循环。
3. SHL/SHR是逻辑左/右移位，大家可以自己去查查，太长了，我就不贴了。
4. SAL/SAR是算数左/右移位，这个跟C语系的移位差不多。

所以下面放出改进后的代码（代码写法很多，下面把一个最通用的写法贴出来，也是wiki908贡献的，我在他的基础上优化了一下下效率，我原来用的是嵌入式汇编，不够通用）：

|  |  |
| --- | --- |
| **1** | unsigned char decrypto( unsigned char s ) |
| **2** | { |
| **3** | return s >> 2 | s << 6; |
| **4** | } |

这也就说明，( s + s % 4 \* 0xFF ) >> 2 等价于 Ror s, 2，微妙的数学关系。

这样，加密也可以同理，就是执行ROL的运算，这里不赘述了，贴一下代码走人。效果和wiki908用密码学知识解密出来的是一样的。

|  |  |
| --- | --- |
| **1** | unsigned char crypto( unsigned char s ) |
| **2** | { |
| **3** | return s << 2 | s >> 6; |
| **4** | } |

下面来介绍一下PNG的加密情况。

本来我是想让wiki908再帮忙分析一下的，因为ExtractData可以提取出来部分图片，可以做密码原文与密文的参考例子，但是后来发现，ExtractData是重编码输出PNG文件的，所以没法通过这样对应的方式再找到一个密码学算法解密，于是只能通过调试手段获取了。

## 六、谈谈任务分配及业内默认规范，以及项目管理

（要谈一下文本的摘要和分配）

## 七、数据的修改

## 八、存档分析及进度统计方法

## 九、双语切换标签的制作

## 十、补丁安装包的制作