前言

在前面两篇文章中（这里，这里）总
结了使用 windbg 和 IDA 找出 cvtres.exe
报错的根本原因，并把一些细节问题弄清楚了。但是还剩下一个小细节没有深究
——

如果启动 cvtres.exe 的时候没有指定全路径，windbg
会报系统找不到指定的文件的错误。当时只是根据经验判断需要使用完整路径，但是为什么只使用文件名不行呢？今天就把这个问题也格尽。

> 说明：
> 写完本文，我犹豫了很久要不要发表出来。因为这个问题其实很简单（在设置 PATH 环境变量时，路径多加了双引号）。但是当时的我真的是当局者迷，完全没意识到这个问题，导致花费了很长时间。process monitor，gflag，IDA，windbg，轮番上场，甚至都调试起 windbg 来了（嗯，你没看错，不是用 windbg 调试，而是调试 windbg），好一阵忙活，中间还走了很多弯路（自以为是的在错误的函数中下断）。最后幡然醒悟，原来真理就在那躺着，静静的等着被发现。
>
> 之所以决定发出来有几点原因：
>
> 介绍了让进程自动中断到调试器中的方法，甚至是让一个调试器自动中断
>
> wt 追踪函数调用的方法等。提醒自己，在调试过程中一定要保持清醒的头脑。
>
> 没耐心的朋友直接跳到最后即可。

问题回顾

相信小伙伴儿们都已经知道可以通过 gflags 设置 Image File Execution Option
来让指定程序启动时自动中
断到调试器。但是当启动 cvtres.exe
的时候，如果只指定文件名，windbg 会报错。报错截图如下：

提示： 注意上图中高亮部分。一切皆因它而起。

难道 windbg 不是通过 CreateProcess()
创建的新进程？看来需要调试一下 windbg 了。但是 windbg
也是自动启动的，该如何调试呢？

螳螂捕蝉，黄雀在后

还记得我们是怎么调试 cvtres.exe 的吗？可以用同样的方法调试 windbg ——通过 gflags 为 windbg 设置 Image File Execution Option。

但是有一个细节需要注意：Debugger 字段需要写一个其它调试器的路径。

cdb 与 windbg 同源，大多数命令与 windbg 一样，是一个非常合适的调试器。

小提示： 可以使用 |
命令查看正在被调试的进程模块名。非常实用的小技巧。

windbg 已经中断下来了，应该在哪里下断点呢？

断错函数了？

创建进程应该会走 CreateProcess() 系列中的一个，猜测 windbg
应该也是调用 CreateProcess()
创建的子进程，与创建普通进程不同
的是，windbg 需要调试新启动的进程。当为 CreateProcess() 函数的
dwCreationFlags 参数设置一个特殊的标志位
DEBUG_PROCESS
时，在进程创建过程中就会建

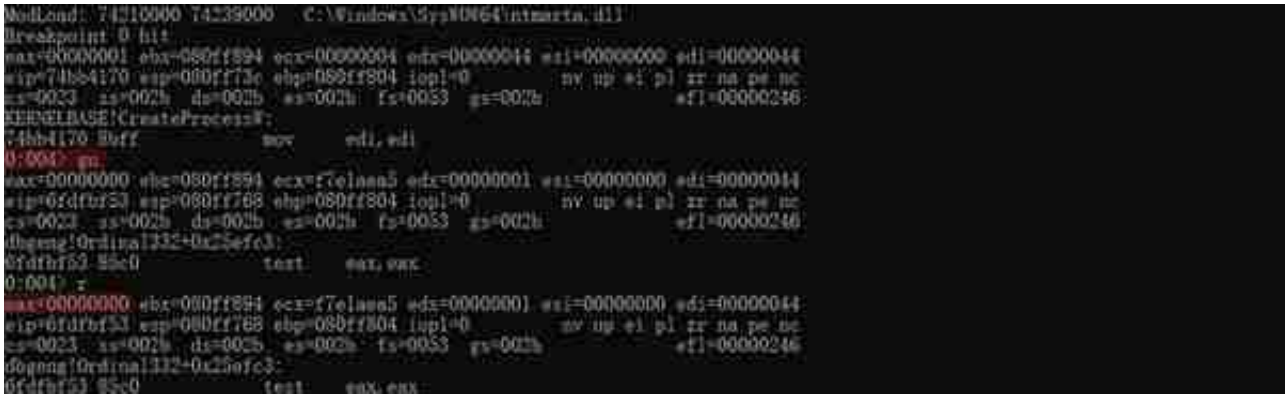
立相关的调试对象，windbg 就能调试对应的进程了。

按照上面的理论，在 cdb 中输入 x
*!*CreateProcess*
查看相关函数，发现
了一堆相关函数。想着最终肯定会调用 ntdll
模块中的函数，于是在 cdb 中输入 bp
ntdll!NtCreateProcessEx，然后执行 g 命令重新运行 windbg。
但是居然没断下来。难道 windbg 不是通过
NtCreateProcessEx()
启动新进程的吗？还是在调用 NtCreateProcessEx() 之前就检查到异常返回了？

对比正常流程

通过全路径启动 cvtres.exe 的时候，windbg
可以正常启动对应的程序。
可以对比查看正常情况下 windbg 调用的是哪些函数。

通过 process
monitor
监听相关事件，只查看进
程相关的事件，可以看到启动 cvtres.exe 时的调用栈，如下图所示：



看来问题出在 kernelbase!CreateProcessW() 函数内部
，用 IDA 看了下 kernelbase!CreateProcessW()
的反汇编代码，发现该函数只是直接
调用了
kernelbase!CreateProcessInternalW()。而
kernelbase!CreateProcessInternalW()

的反汇编代码超级多，即使用 IDA 的 F5
得到的伪代码都非常多，实在没耐心看完。那该怎么办呢？脑子中突然想起一个很久之前就知道但是一直没真正使用过的命令 —— wt。

wt 立功了

重新运行程序，当 windbg 中断到 cdb 后，输入 bp kernelbase!CreateProcessInternalW 设置好断点，输入 g 重新运行 windbg。中断下来后，输入 wt -l1 只追踪一层调用。

> 提示： 因为 wt 的输出结果很可能会有很多，所以最好先执行 .logopen d:\windbg.txt 来打开日志文件，追踪完成后再执行 .logclose 关闭日志文件，这样就可以直接查看日志文件中的追踪结果了。

查看日志文件中的调用记录，很快就发现了让我头脑瞬间清醒的几个函数调用。一个是 SearchPathW 一个是 RtlSetLastWin32Error。

修复

既然是指定的路径出问题了（不能带双引号），修改脚本内容如下：

```
12
```

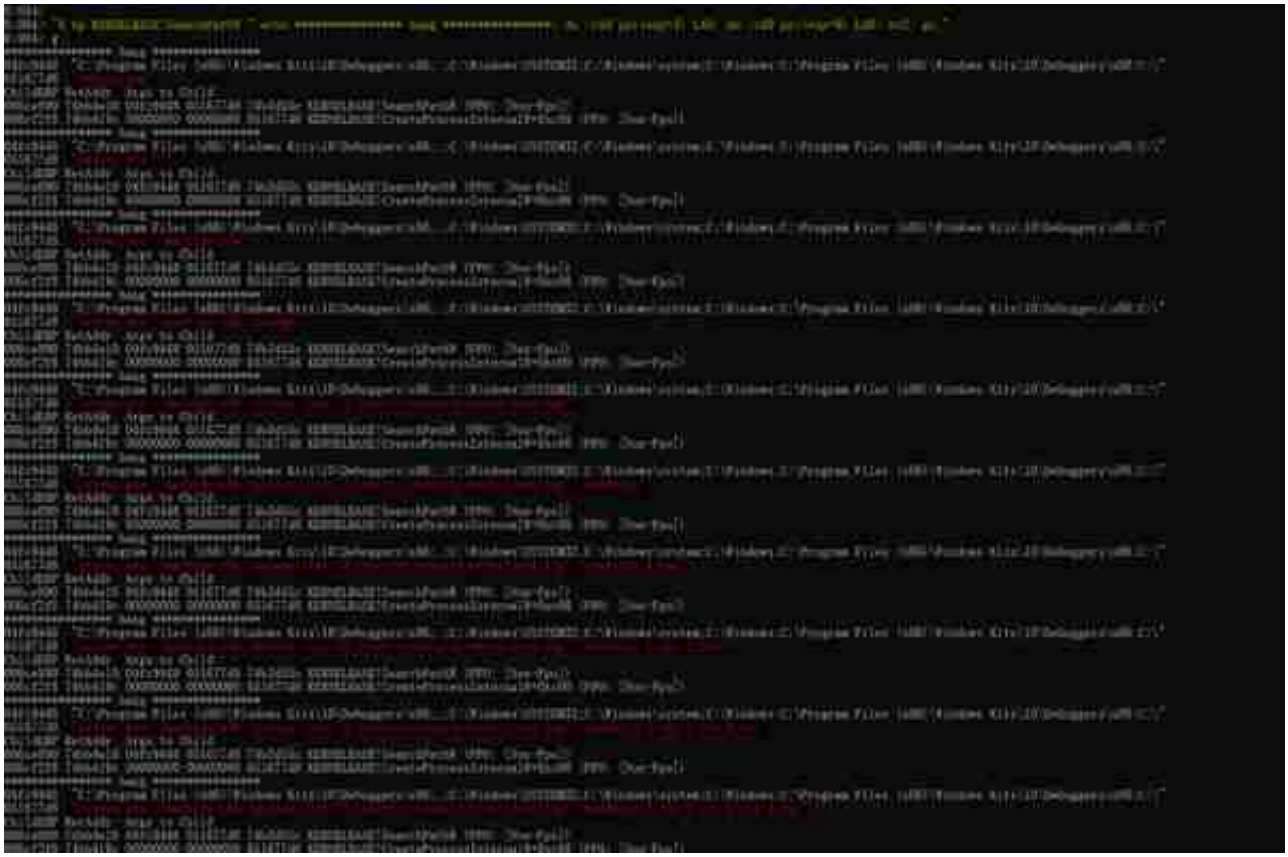
copy

```
set path=%path%;C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\cvtres.exe /machine:x86 /nologo /out:"d:\test-rc-compile-error\test.tmp" /readonly 1.res 2.res 3.res 4.res 5.res
```

再次运行脚本，果然一切正常了。

有趣的访问方式

当我在 process monitor 中修改过滤条件后，我还看到了下图的访问记录：



反思

其实这个问题本来应该很快就能解决的。

首先，在调试之前就应该使用 process monitor 查看文件访问事件。只不过当时脑子不清醒，只是用 process monitor 查看了进程创建事件，没有什么发现就开始调试了，太草率了。

其次，当时陷入了思维误区 —— 想当然的认为带空格的路径一定要加上双引号。整个排查过程中都没有怀疑是多加了双引号导致的问题。反过头来看这个问题，觉得自己太傻了，查看 PATH 环境变量的时候，非常明显可以发现其它带空格的路径并没有使用双引号，但是我自己添加的路径却带了双引号。

收获

- 长记性了，设置 PATH 环境变量的时候，即使路径中包含空格也不能加双引号。
- 再次熟悉了使用 gflags 设置 Image File Execution Option 的方法，这次是使用调试器调试另外一个调试器。
- 熟悉了 wt 的用法。很早之前就知道 wt 的存在，一直没实际用过。当想查看一个函数都调用了哪些子函数时，非常方便。
- 熟悉了为指定线程设置断点的方法。语法很简单，很容易记忆。在 bp 之前添加 ~tid，即可为指定的线程设置断点。
- 知道了突破默认输出列宽的方法 —— 通过 /c 指定显示列宽。
- 熟悉了 CreateProcess 的部分逻辑，创建进程真的是个大工程。

参考资料

https://docs.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw

https://docs.microsoft.com/en-us/windows/win32/procthread/process-creation-flags

https://docs.microsoft.com/en-us/windows/win32/debug/system-error-codes--0-499-

https://docs.microsoft.com/en-us/windows/win32/api/processenv/nf-processenv-searchpathw

https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/d--da--db--dc--dd--dd--df--dp--dq--du--dw--dw--dyb--dyd--display-memor