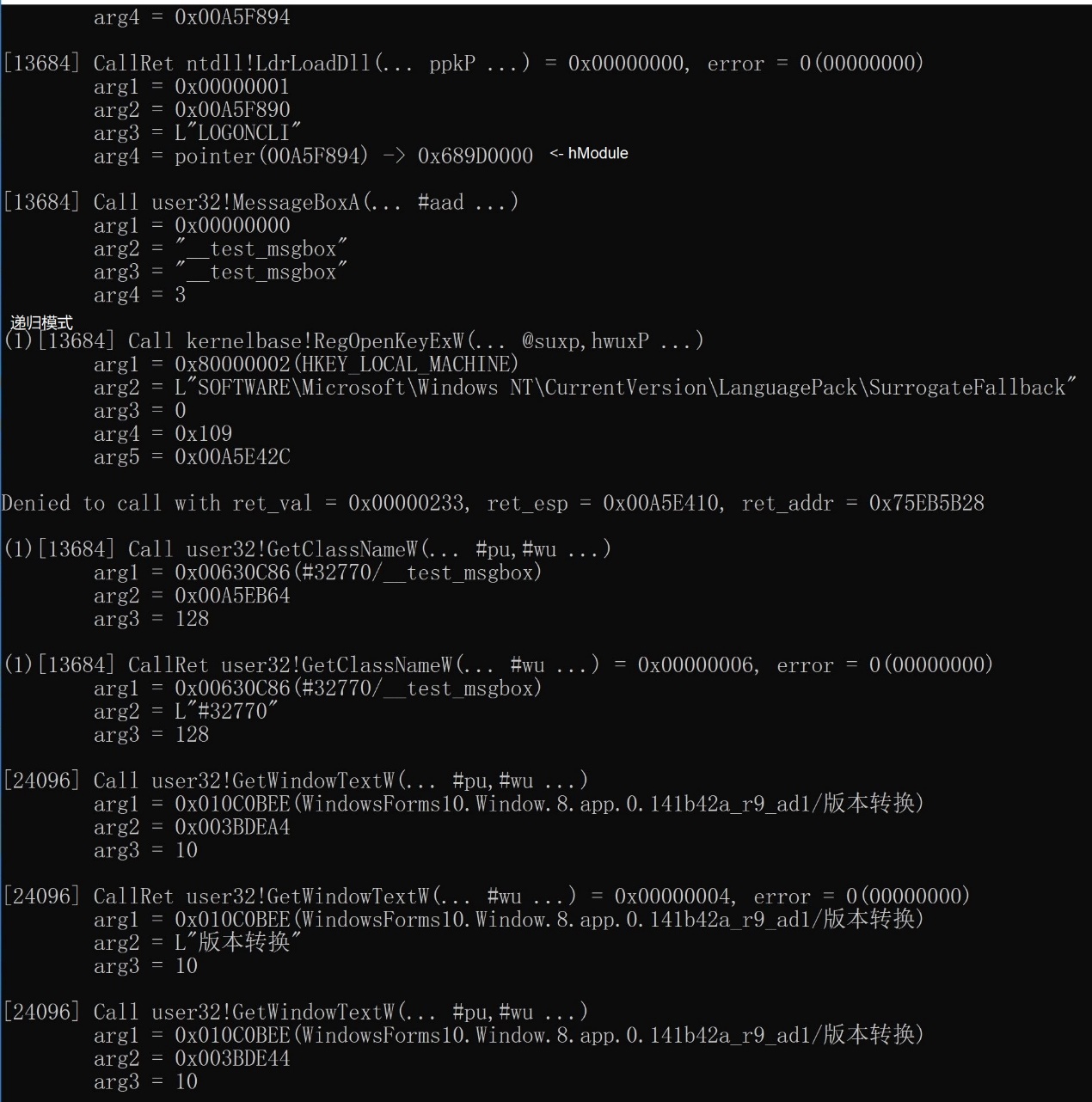
**Win32Exts\_for\_API\_Monitor**

**Win32Exts for API\_Monitor** 是Win32Exts项目组提供的一个强大的API 监视工具，相较于流行的 API\_Monitor，它具有解码功能强大、灵活二次扩展开发、与其他脚本语言完美融合交互 等等优异的特性。

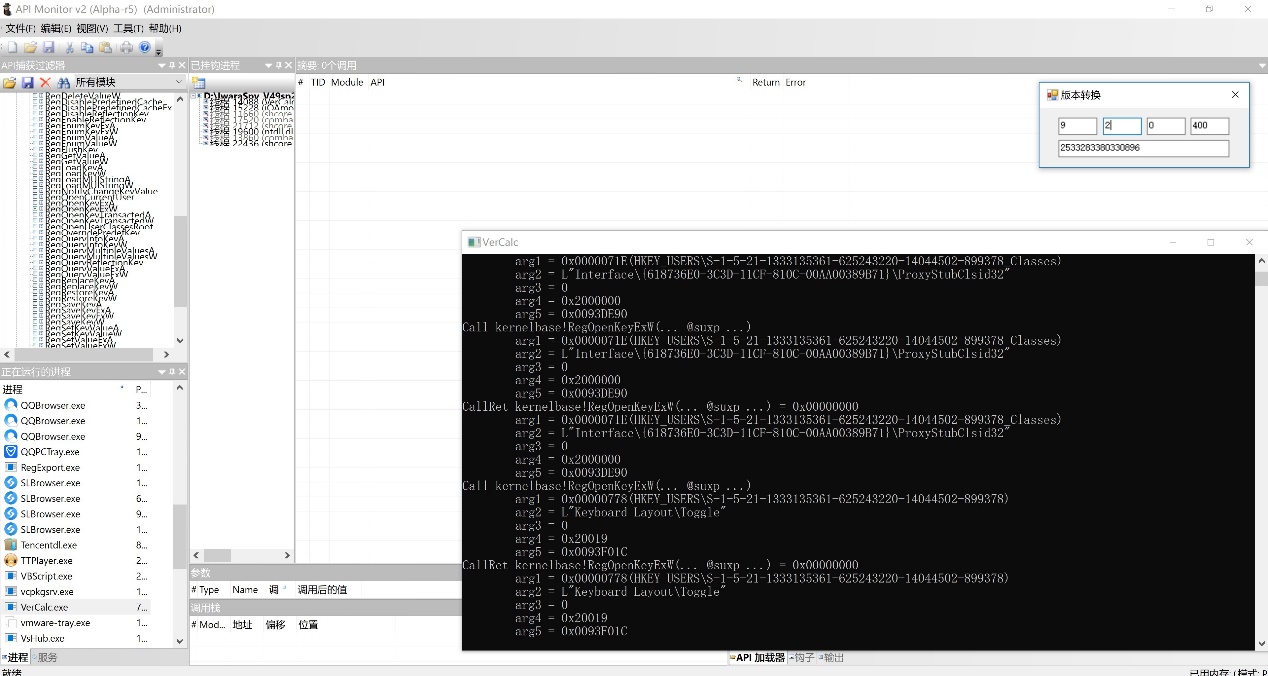


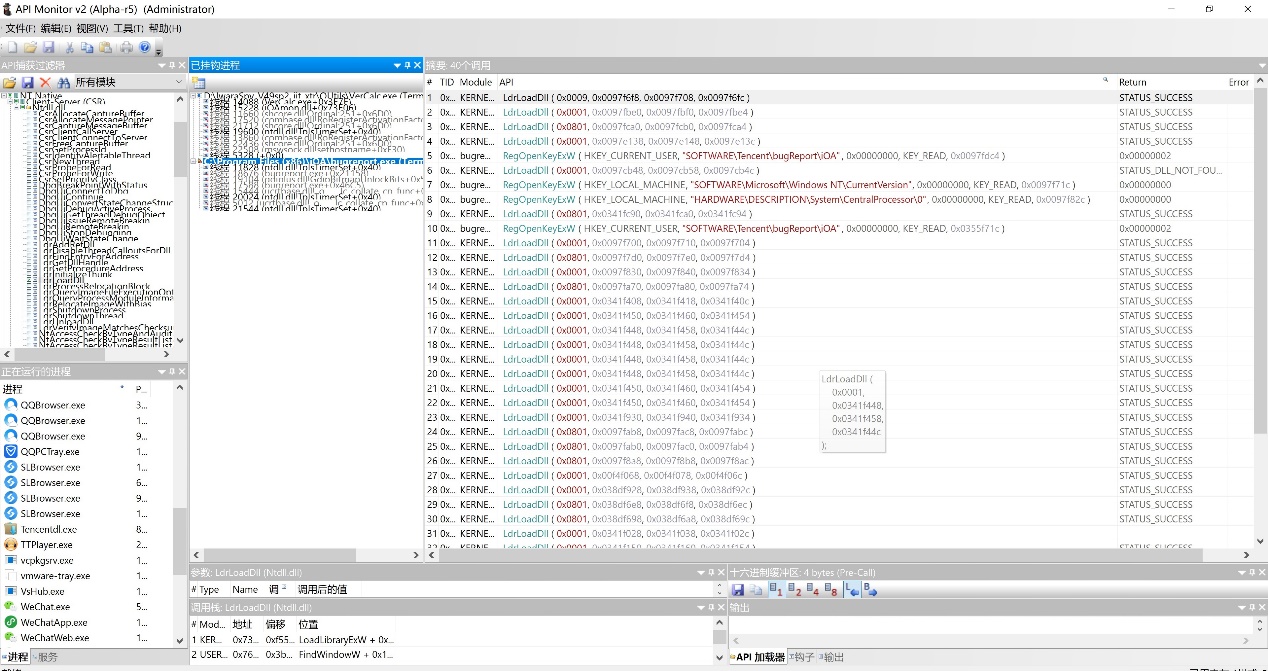
Github下载地址：

<https://github.com/tankaishuai/Win32Exts_for_API_Monitor>

编者按：

开发此项目的目的是为了弥补当下流行的 API Monitor 的一些不足之处：

1. ***很多时候API无法成功监控***，例如笔者在 Win10 x64 管理员权限下监控 API **RegOpenKeyExW**，API Monitor 并没有正确的工作。相反， Win32Exts\_for\_API\_Monitor 却工作的很好。
2. ***API Monitor 的解码参数信息过于简单***，通常仅支持解码一些整型参数与字符串，以及常量表示。Win32Exts\_for\_API\_Monitor 却可以支持解码各种复杂的数据类型以及格式表示，同时对于预定义不支持的类型也提供了插件扩展方式支持之。

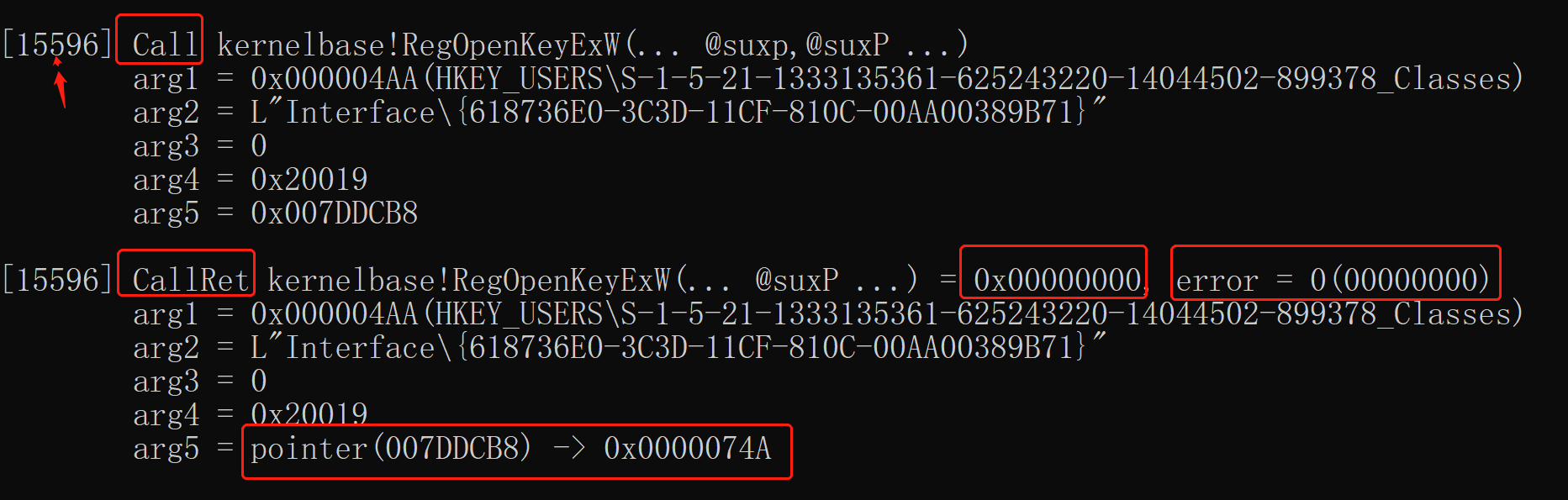


1. ***API Monitor 仅仅只能监控API函数调用完成之前的结果***，对于调用之后的信息并不能正确地得到。例如以下形式的导出函数：

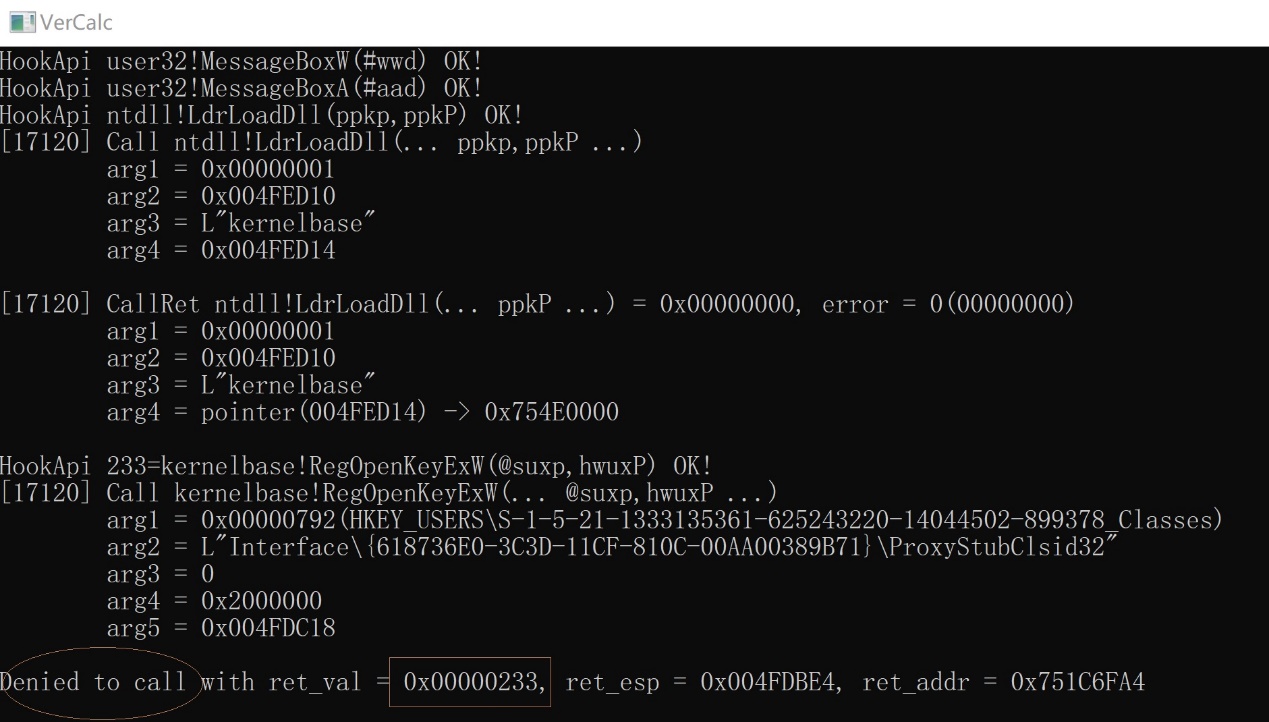
BOOL **PathAppendW**(\_\_inout WCHAR \*pszPath, LPCWSTR szSubPath)；

由于参数 pszPath 是一个**输入输出型**参数，API Monitor 并不能正确拿到函数调用之后pszPath是什么数据内容。

Win32Exts\_for\_API\_Monitor 可以正确监控API调用之前，以及调用之后的参数、返回值 信息。同时线程信息、LastError信息也一应俱全。



(4) **API Monitor 仅仅只能监控API 调用，不能阻止这种行为**。**Win32Exts\_for\_API\_Monitor 可以实现阻止指定的API调用，并且伪造一个指定的返回码 以返回调用上下文。**例如 以下阻止 RegOpenKeyExW 直接返回 0x233：



当然，API Monitor 也有一些优点是 Win32Exts\_for\_API\_Monitor所不具备的，具体体现在：

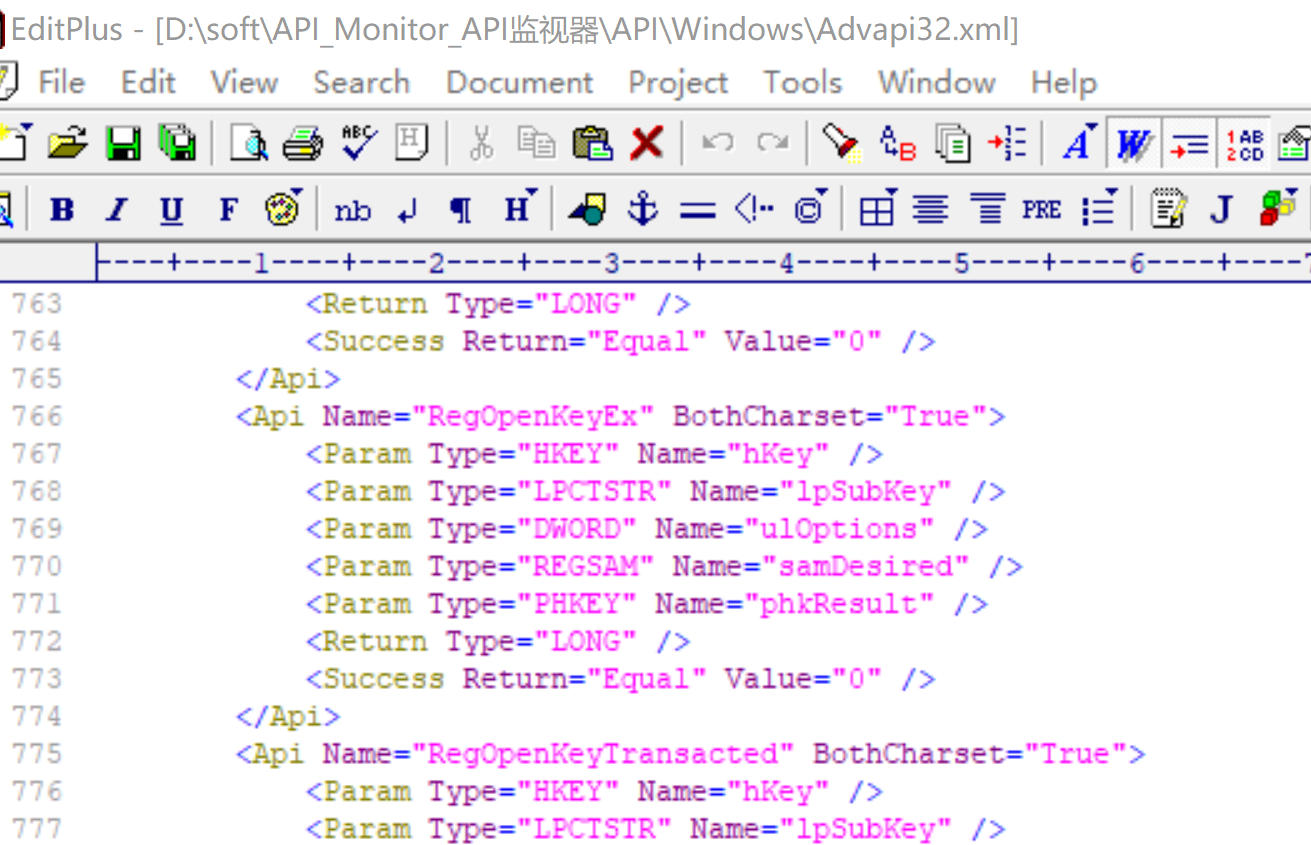
1. ***预定义了大量的API声明，但也限制了其灵活性***。 Win32Exts\_for\_API\_Monitor 并不提供除示例外的任何API声明， 可以非常灵活的按照输出格式自由配置。
2. ***友好的用户界面***，目前 Win32Exts\_for\_API\_Monitor 并不提供一个完整的用户界面，捕获的函数调用信息 以控制台、日志、或者调试输出的方式记录。

鉴于此，大家可以酌情选择使用之。

本文主要介绍 **Win32Exts\_for\_API\_Monitor** 的基本用法：

1. **设置抓取API列表**

API Monitor 的API配置是以XML形式保存的，例如：



比较复杂，Win32Exts\_for\_API\_Monitor 仅仅只需要在**win32exts.ini** 中按如下格式记录参数类型符即可：

**index=[fake\_ret\_val=或\*]module!function,[^][in\_args][,out\_args]**

或

**index=[fake\_ret\_val=或\*]module+offset,[^][in\_args][,out\_args]**

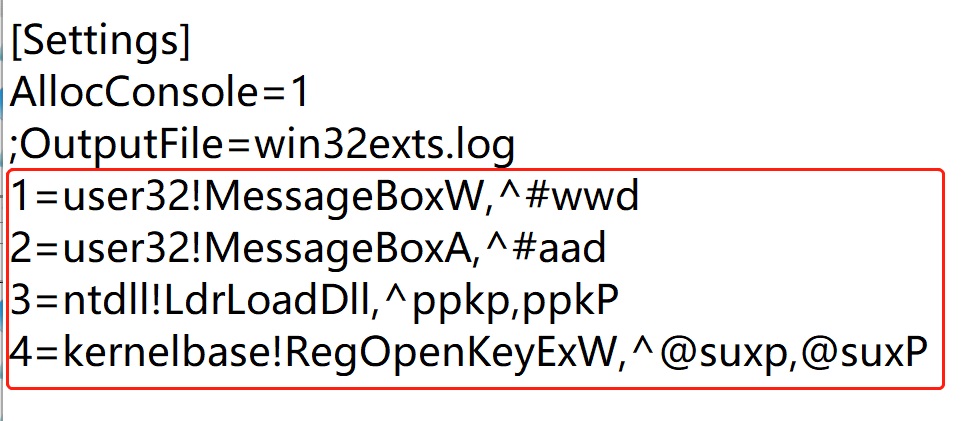
或

**index=[fake\_ret\_val=或\*]virtual\_addr,[^][in\_args][,out\_args]**

其中 [] 表示可选参数，

**^表示监控函数返回，如果不指定out\_args，则缺省与in\_args类型描述相同。**

**如果指定 [fake\_ret\_val=或\*] 这个，表示拦截该API的调用（\_\_stdcall 用 =，\_\_cdecl 用 \*），并以 fake\_ret\_val 作为返回值返回调用上下文。**



以上配置中逗号（,）后面的 **in\_args、out\_args**是参数类型描述，预定义的类型描述见下表：

==================================================

a : LPSTR (gbk)

**t : LPSTR (utf-8)**

w或s : LPWSTR

i或d : int

u : unsigned int

x : hex

b : true/false

c : char

h : short

l : int64

f : float

g : double

v : VARIANT \*

k : UNICODE\_STRING \*

m : BSTR

p : hex

@ : 注册表 hkey

# : 窗口 hwnd

$ ： 文件handle

| : 进程 handle

! : 进程 id

~ ：线程 handle

. : 模块 hDll

< : 局部atom

> : 全局atom

? : IUnknown \*

& : 二进制（buf, size）

% : 二进制(buf)

‘ : time\_t

“ : FILETIME \*

==================================================

另外，**字母大小写有区别，小写表示按值传递，而大写表示按引用方式传递**。例如监控：

1=psapi!GetModuleFileNameExW,|pwu

会输出完整的 进程名、模块句柄、模块名、缓冲区尺寸。

配合脚本语言，可以支持运行时动态的添加、修改该API监控列表。

1. **开始监控进程**

**执行批处理**：

**Api\_Monitor** 16进制的进程窗口句柄

或者**直接运行**：

**rundll32** "%cd%\**win32exts.dll**",**MyAnyHookApi\_RunDll32** --hwnd=16进制的进程窗口句柄

即可开始监控。

对于**无窗口进程，则可以使用下列命令行**：

**rundll32** "%cd%\**win32exts.dll**",**MyAnyHookApi\_RunDll32** –thread=目标线程Id –type=钩子类型

1. **与Lua等脚本的交互**

由于 **Win32Exts for Lua/JavaScript/VBScript/Python**等等过于复杂，可实现的功能非常丰富，需要了解的可联系作者（tankaishuai@126.com）沟通。

例如以下Lua脚本直接在目标进程注入一个扩展模块：



1. **扩展插件的编写**

编写一个常规 Dll：MyAnyHookDisplayCall.dll 实现并导出下列接口：

bool **MyAnyHookDisplayCall**(

LPCSTR szFuncName,

bool bIsCallRet,

*INT\_PTR* \*pArgs,

LPCSTR \*pszFmt,

*\_\_inout* LPSTR \*lpszOutText){

… …

}

即可实现自定义的参数类型解码输出。如有需要，具体开发细节可联系作者。