【分析Windows源码】【绕过360加载驱动检测】【结束360关键进程】

2018年4月7日

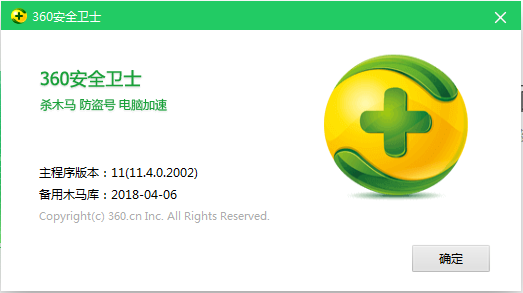
8:55

**环境：**Win7 7600 x86

**360版本：**11.4.0.2002

**目标1：**绕过360加载驱动检测。

**目标2：**实现结束360关键进程。



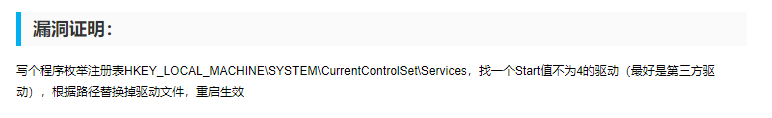
# **一、Ring3绕过360加载驱动检测**

## **1.1、选取方向**

通过一个别人提交的漏洞，虽然厂家已确认，但抱着侥幸的心里测试下。

**【绕过360安全卫士任意加载驱动（绕过主动防御）】**

<https://www.secpulse.com/archives/19400.html>



## 

## **1.2、确定寄宿者**

选取第三方驱动，要找一个几乎每个电脑上都有的驱动。

为什么选它呢？因为它是Tencent驱动加载顺序最后的。

(这里节省时间，后面代码省略枚举注册表这块代码，指定固定软件的固定路径)

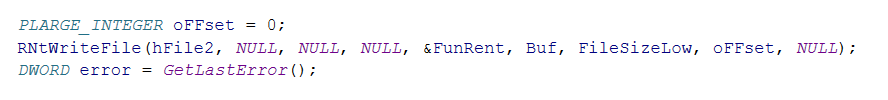
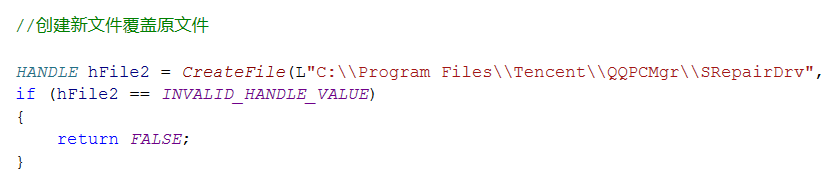


## 

## **1.3、Test**

360确实对相关函数进行检测拦截，推测是HOOK。

手写代码调用**CreateFile** 和 **ntdll**的**NtWriteFile**函数。



经测试，通过**CreateFile**成功覆盖文件，但

360在**Ring0**对**NtWriteFile**进行了拦截检测。



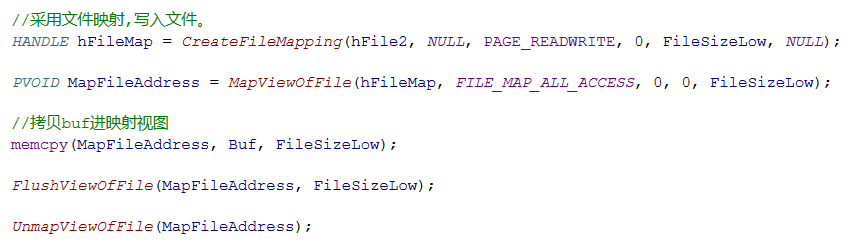
## **1.4、移花接木**

这样就结束了吗，不是的。

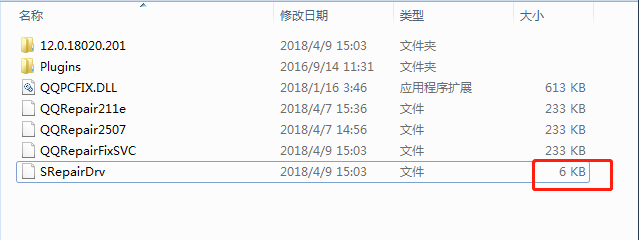
覆盖文件成功，但写入文件被拦截。

写入文件的方式其实还有一种：**通过文件映射对象。**

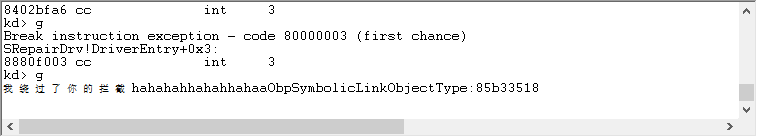
**文件映射：**



**成功映射：**



**重启后成功加载驱动：**



## **1.5、Extend**

这是建立在Win7 32上的。

在x64 和 Win10 上，可以尝试手工或一些工具绕过PG保护和数字签名。

后面还有内容呢，所以本篇文章就不做x64 和 Win10的Test了。

**另：已向360安全应急响应中心提交该漏洞。**



# **二、实现结束360关键进程**

最开始是研究A-Protect的源码，但发现很多功能都是仿造WRK的，所以直接结合Windows源码看，还是Windows源码强大压，所以下方会结合WRK和win2000和A-Protect的源码进行分析。

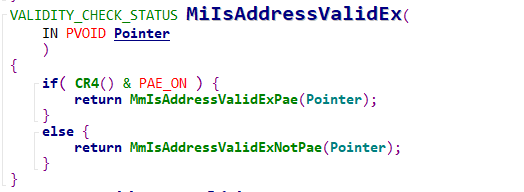
主要实现功能是结束360进程。

## **1、MiIsAddressValidEx**

先介绍一个配角，一个判断地址是否有效的函数，下面一些地方会用到它。

函数的内部实现是基于Windows源码的**MiIsAddressValid**的，基本实现很类似，多加了一些内部判断，如果对这一块不熟悉的，可以先看后面的3.Windows源码解析。

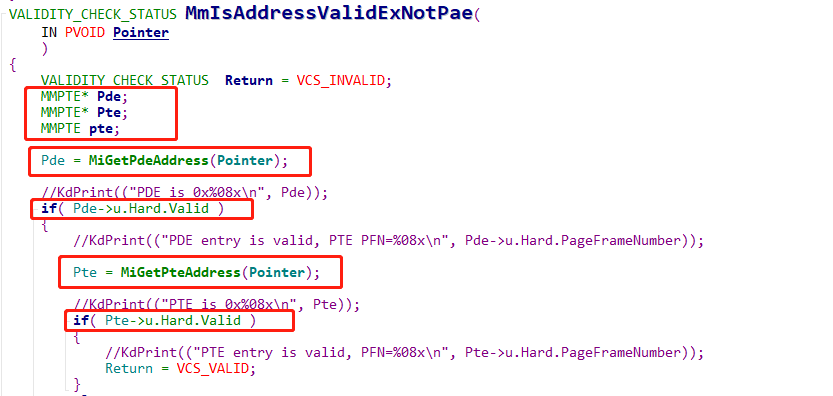
大牛直接往下看~



做了是否开启PAE的判断，并调用对应的函数。

未开启PAE的函数**MmIsAddressValidExNotPae** 和 开启PAE的MmIsAddressValidExPae雷同。

很多东西跟Windows的MmIsAddressValid是一样的，这里就不再赘述了。

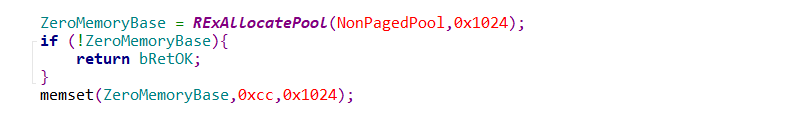


## **2.1、ZeroProcessMemory**

功能：破环进程空间。

关键代码：

申请一段空间，并填充0xCC。



先用**MiIsAddressValid**验证页面地址是否有效，再用**ProbeForWrite**验证用户内存是否可写，再对进程空间进行填充0xCC。



## **2.2、杀掉进程其它实现：**

这里使用破环进程内存就能结束进程了。

A-Protect中提到使用PspTerminateThreadByPointer强制结束进程线程。

但经测试，会蓝屏，不能武断地用PspTerminateThreadByPointer结束。

况且如果PspTerminateThreadByPointer被HOOK拦截了呢？

后续会发一篇结束进程的几种方式。

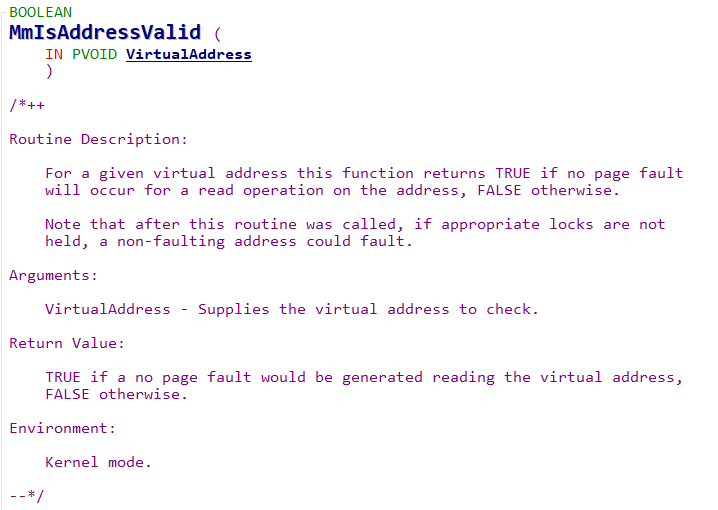
## **3、分析Win2000源码**

下面是对**MmIsAddressValid、ProbeForRead、ProbeForWrite**的源码分析。

因为上面A-Protect部分源码 其实就是用到了Windows的源码逻辑。

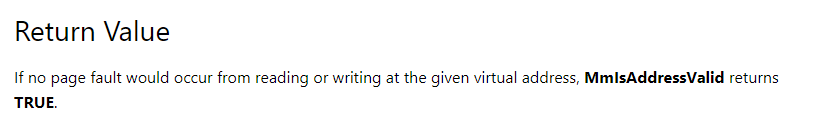
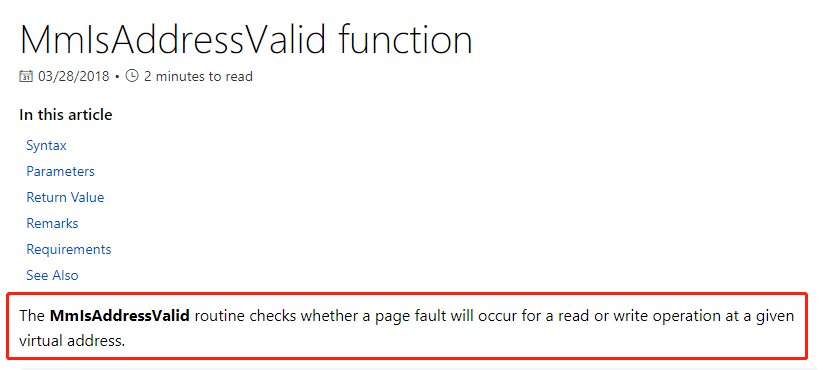
大牛这部分可绕过~

### **3.1、MmIsAddressValid**

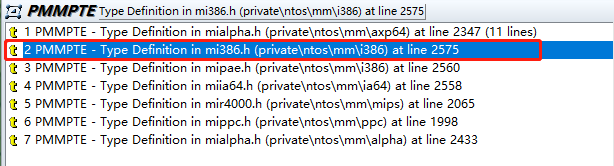


**附：**

**MSDN文档：**



**以下分析的 结构体 和 宏 均采用32位下的：**

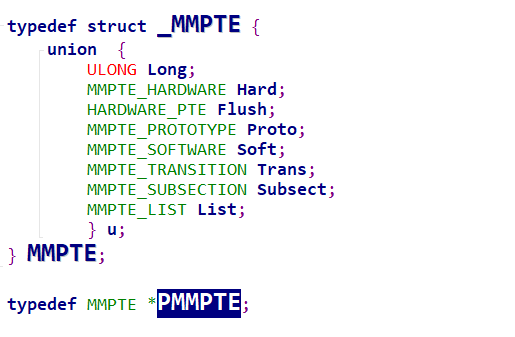


#### **3.1.1、定义了一个结构变量指针，**

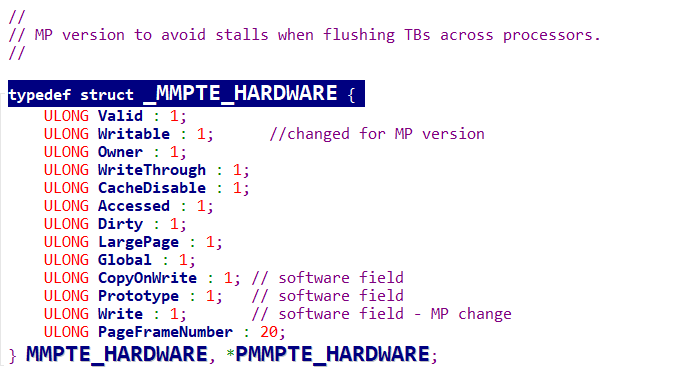


**附：**

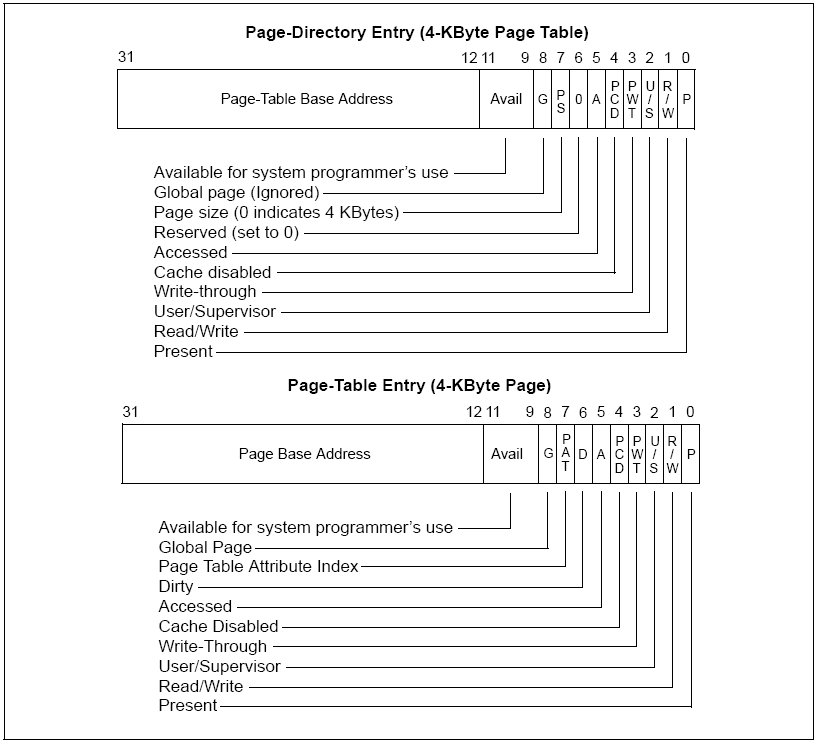
**32位模式下的结构体，后面重点用到Hard字段。**



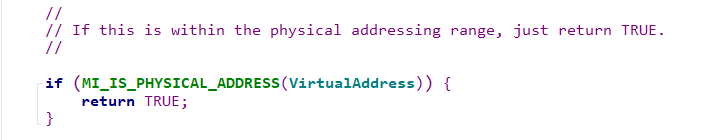
**查看Hard字段类型。**



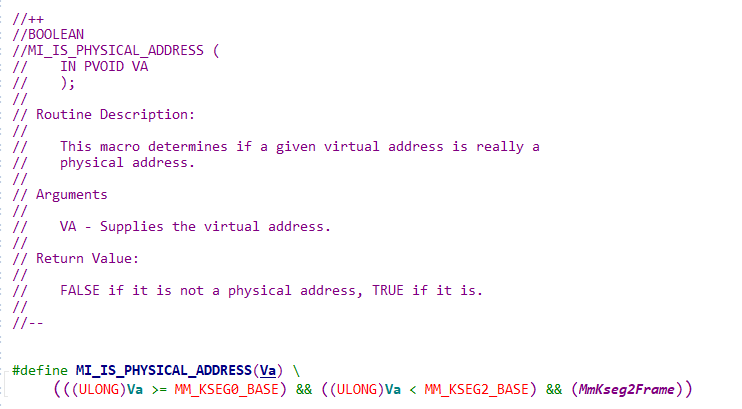
**其实就是类似 PDE 和 PTE 的共用联合体。**



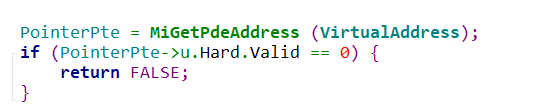
#### **3.1.2、判断给定地址是否为物理地址，是则该地址有效。**



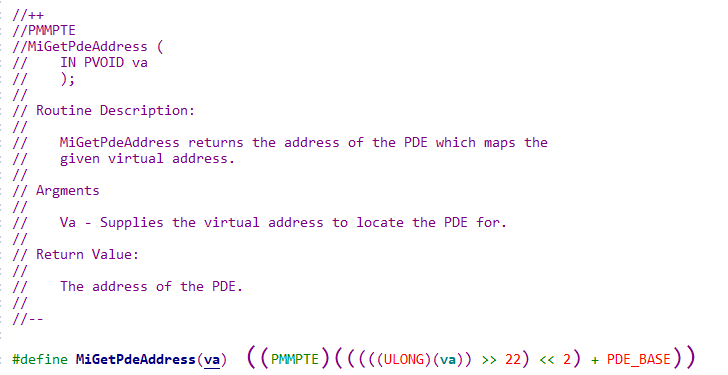
**附：**



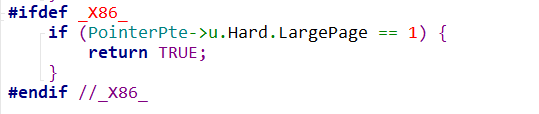
#### **3.1.3、根据虚拟地址得到PDE，并验证PDE的P位是否有效，无效返回FALSE。**



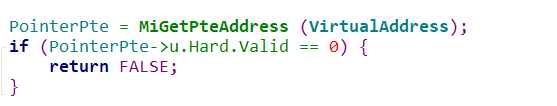
**附：**



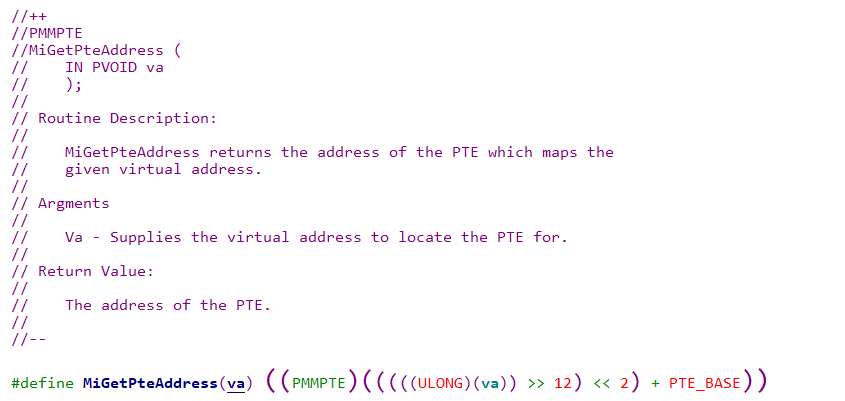
#### **3.1.4、x86检测PDE的P/S位，是否为4MB的大页(PDE直接指向物理地址)。**



#### **3.1.5、根据虚拟地址得到PTE，并验证PTE的P位是否有效，无效返回FALSE。**

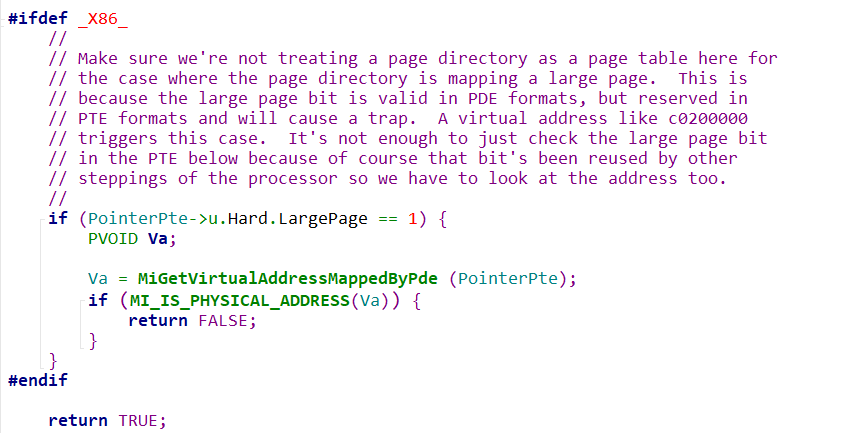


**附：**

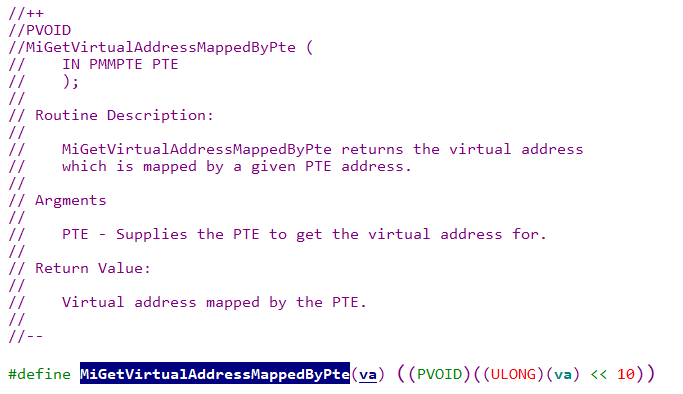
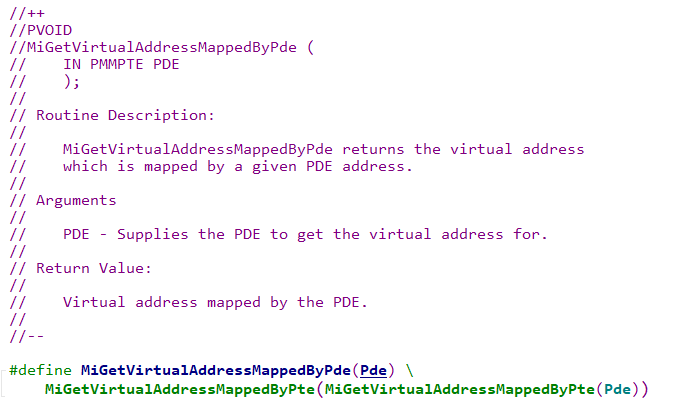


#### **3.1.6、P/S位只在PDE有效，防止PTE被当作PDE，映射到大页面，对PTE进行检查。**

**(如果PTE的P/S位有效，则把PTE当PDE检测，如果返回的物理地址有效，则PTE被错当成PDE了。返回FALSE)**



**附：**



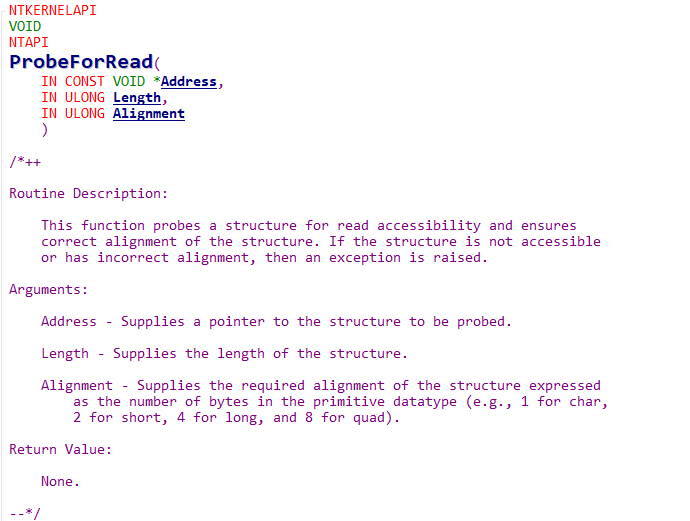
#### **3.1.7、总结：**

**由以上可见MmIsAddressValid只是检测一下是否为物理地址，PDE或PTE的标志位P位是否有效，并未对页面是否可读可写进行检验，真正起作用的其实是下面的 ProbeForWrite。**

### 

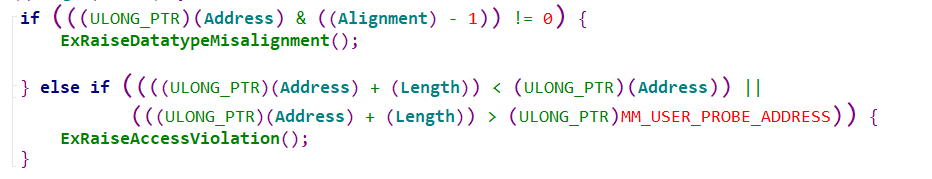
### 

### **3.2、ProbeForRead()**

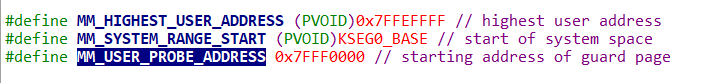


不知是源码缺失 还是ProbeForRead本身就不做读的检测。

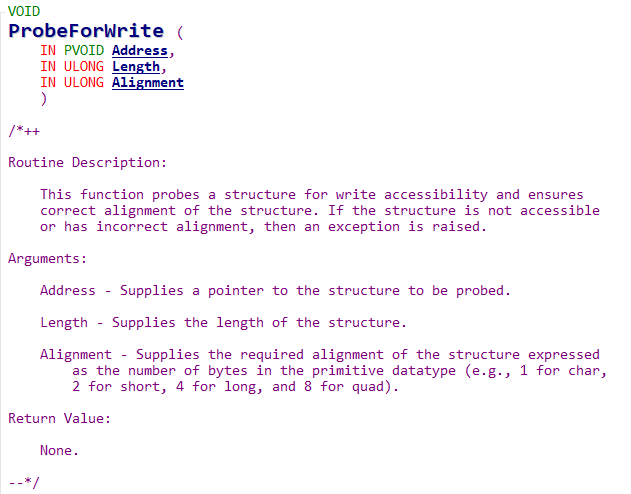
以下代码只是做了下常规判断，否则就抛出异常，就没有然后了。



附：



### **3.3、ProbeForWrite()**



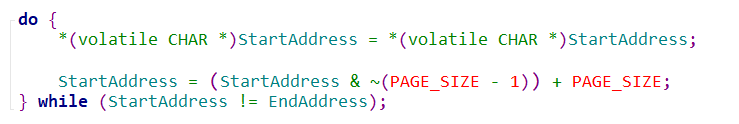
StartAddress == 传入地址。

EndAddress == 页面末尾 (如果跨页，则为下页末尾)。



以 \*(volatile CHAR \*)StartAddress = \*(volatile CHAR \*)StartAddress 方式进行验证读写，如果一个字节可读写，则此页面可读写。

并以 StartAddress = (StartAddress & ~(PAGE\_SIZE - 1)) + PAGE\_SIZE 进行页面移动。



### **3.4、总结：**

**MmIsAddressValid根据PDE PTE的P位判断页面是否有效。**

**ProbeForWrite会检测页面是否可读写。**

**通过Windows源码可实现许多骚操作。**

# **三、实现代码：**

## **1、绕过检测代码**

见附带文件。

## 

## **2、结束进程代码**



效果图：

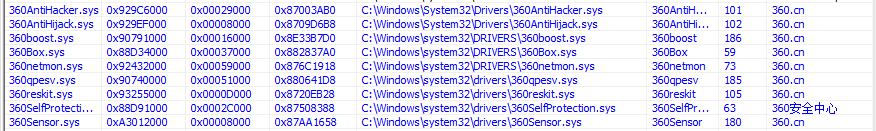


360内部多进程，且加载了许多驱动，结束一段时间后，Trap进程恢复，推测内部有进程守护检测。

再次使用ZeroProcessMemory失效，这时就必须使用高级一点的玩法了。

（其它进程的结束方式会在后续文章）

360驱动：。。。



见附带文件。

***KID***