

## 前置知识：

大部分驱动除了需要具备读写设备的能力，还需要具备对硬件控制的能力

1. 在用户空间，使用*ioctl*系统调用来控制设备，原型如下：

```
1 int ioctl (int fd, unsigned long cmd, ...)
2 //fd:文件描述符
3 //cmd: 控制命令
4 //....:可选参数：插入*argp，具体内容依赖于cmd
```

用户程序做的只是通过命令码告诉驱动程序它想做什么，至于怎么解释这些命令和怎么实现这些命令，这都是驱动程序要做的事情

2. 驱动*ioctl*方法：

```
1 int(*ioctl)(struct inode *inode, struct
file*filp,unsigned int cmd,unsigned long
arg);
```

- 2 //inode和filp两个指针对应于应用程序传递的文件描述符fd，这和传递open方法的参数一样
- 3 //cmd由用户空间直接不经修改的传递给驱动程序
- 4 //arg可选

在驱动程序中实现的ioctl函数体内，实际上是一个switch（case）结构，每一个case对应一个命令码，做出一些相应的操作

## 参数介绍：

"-l": 开启日志记录模式（不会影响主日志记录模块）

"-s": 驱动枚举模块

"-d": 打开设备驱动的名称

"-i": 待fuzz的ioctl code，默认从0xn timer 0000-

0xn timer ffff

"-n": 在待探测阶段采用null pointer模式，该模式下极易fuzz到空指针引用漏洞，不加则是常规探测模块

"-r": 指定明确的ioctl code范围

"-u": 只fuzz-i参数给定的ioctl code

"-f": 在探测阶段采用0x00填充缓冲区

"-q": 在fuzz阶段不显示填充input buffer的数据内容

"-e": 在探测和fuzz阶段打印错误信息

"-h": 帮助信息

常用的Fuzz命令实例:

```
1 BirdFuzzer.exe -s
```

进行驱动枚举，将CreateFile成功的驱动设备名称，以及部分受限制的驱动设备名称打印并写入Enum\_Driver\_log.txt文件中

```
Enum_Driver_log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Validate Driver: HarddiskVolumeShadowCopy2
Validate Driver: fsWrap
Validate Driver: HarddiskVolumeShadowCopy3
Validate Driver: HarddiskVolumeShadowCopy4
Validate Driver: Secdrv
Validate Driver: HarddiskVolumeShadowCopy5
Validate Driver: COMMU
Validate Driver: NxtIPSecDevice [But no privilege, check if Administrator, or try another Driver]
Validate Driver: {C8456DB2-55E6-4C56-B780-0730E61135C4}
Validate Driver: HarddiskVolumeShadowCopy6
Validate Driver: SstpDrv [But no privilege, check if Administrator, or try another Driver]
Validate Driver: DISPLAY4 [But no privilege, check if Administrator, or try another Driver]
Validate Driver: {29898C9D-B0A4-4FEF-BDB6-57A562022CEE}
Validate Driver: {DF4A9D2C-8742-4EB1-8703-D395C4183F33}
Validate Driver: HarddiskVolumeShadowCopy7
Validate Driver: WFPDev [But no privilege, check if Administrator, or try another Driver]
Validate Driver: {71F897D7-EB7C-4D8D-89DB-AC80D9DD2270}
Validate Driver: ProcessManagement [But no privilege, check if Administrator, or try another Driver]
Validate Driver: WfpAie [But no privilege, check if Administrator, or try another Driver]
Validate Driver: MpsDevice [But no privilege, check if Administrator, or try another Driver]
Validate Driver: KJSelfProtectEx
Validate Driver: KJnetmon
Validate Driver: NDIS
Validate Driver: PartmgrControl
Validate Driver: QMCCuber [But no privilege, check if Administrator, or try another Driver]
Validate Driver: PIPE
Validate Driver: ksapi64_dev
```

```
1 BirdFuzzer.exe -d XXX -i 0xaabb0000 -f -l
```

对XXX驱动的ioctl code 0xaabb0000-0xaabbffff范围进行探测，以及对可用的ioctl code进行fuzz，探测时除了正常探测外增加0x00填充缓冲区探测，开启数据日志记录（比如增加-u参数，则只对ioctl code 0xaabb0000探测，若是有效的ioctl code则进入fuzz阶段）

```
[~] Open handle to the device \\.\FltMgr ...
OK

Summary
-----
IOCTL scanning mode      : Range mode 0x00222002 - 0x00222004
Filter mode              : Filter codes that return true for all buffer sizes
Symbolic Device Name     : \\.\FltMgr
Device handle            : 0x00000034

[~] Bruteforce function code + transfer type and determine input sizes...
[!] Write Detect info in logger_detect...Check it!
```

```
1 BirdFuzzer.exe -d XXX -r 0xaabb1122-  
0xaabb3344 -n -l
```

对XXX驱动的ioctl code 0xaabb1122-0xaabb3344范围进行探测，探测时采用null pointer模式，并进行数据日志记录

```

D:\工具包\代码战争\BirdFuzzer\Debug>BirdFuzzer.exe -d FltMgr -r 222002-222004 -n
-1

  _ _ _ _ _
<_>  _ _ _ _ _  _ _ _ _ _  /_>
_ _ _ _ _  _ _ _ _ _  _ _ _ _ _  _ _ _ _ _
_ _ _ _ _  /_>  /_>  _ _ _ _ _  _ _ _ _ _  <_>  _ _ _ _ _
_ _ _ _ _  <_>  <_>  _ _ _ _ _  _ _ _ _ _  >_>  _ _ _ _ _
_ _ _ _ _  _ _ _ _ _  _ _ _ _ _  _ _ _ _ _  _ _ _ _ _  Framework v1.0
Author: @koutoo
Github: https://github.com/koutto/ioctlbf
Customized by k0sh1 @KeyZ3r0
Github: https://github.com/k0keoyo
Contact me: k0pwn_0110@sina.cn

[~] Open handle to the device \\.\FltMgr ...
OK

Summary
-----
IOCTL scanning mode      : Range mode 0x00222002 - 0x00222004
Filter mode              : Filter disabled
Symbolic Device Name    : \\.\FltMgr
Device handle            : 0x00000034

[~] Bruteforce function code + transfer type and determine input sizes...
[!] Write Detect info in logger_detect...Check it!

```

## 日志文件：

log: 主日志记录文件

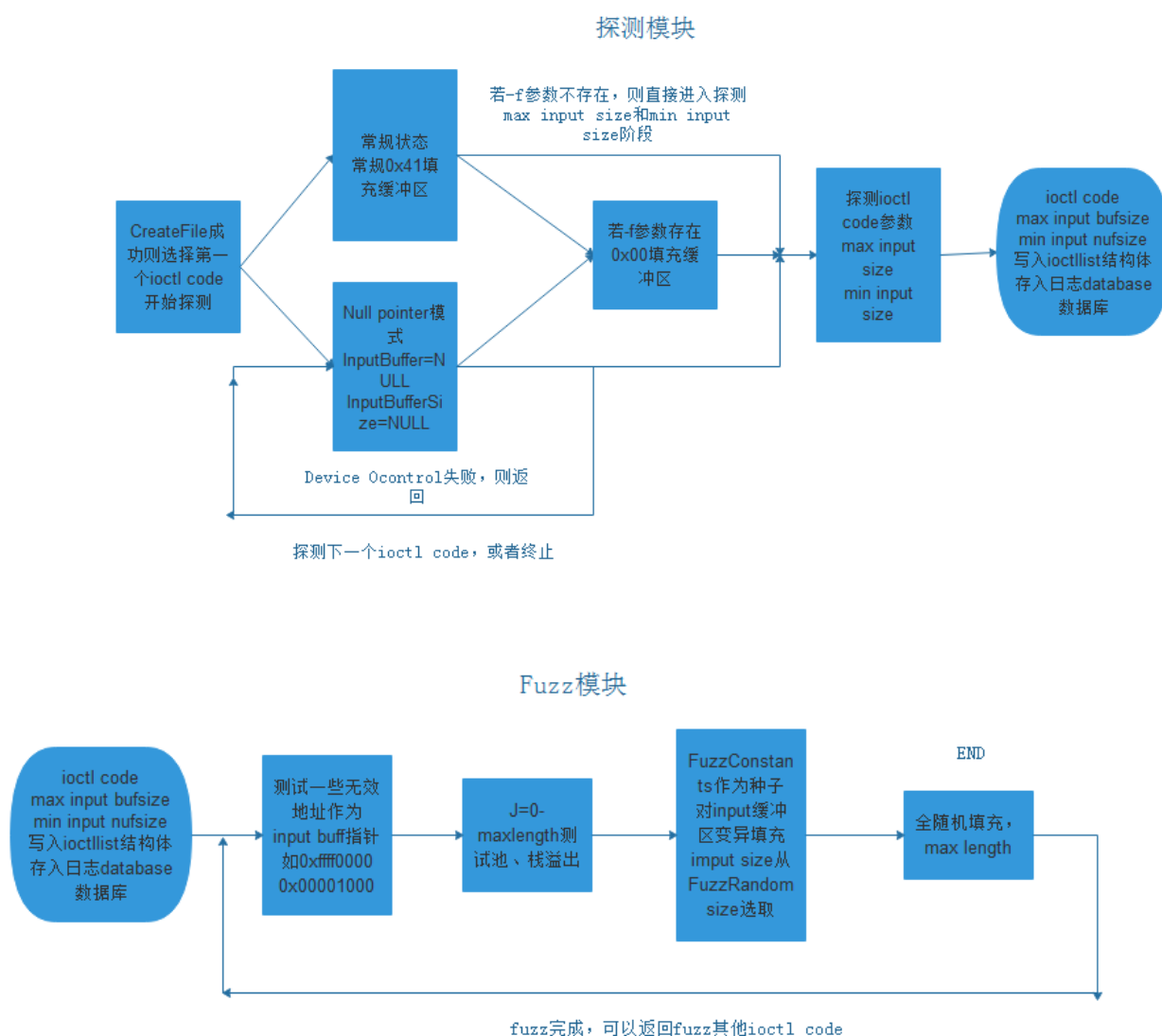
Enum\_Driver.txt:驱动枚举记录文件

log\_detect:探测模块日志记录文件

log\_fuzz:fuzz模块日志记录文件

log\_database: ioctl list数据库记录模块

BirdFuzzer 整体框架和Fuzz思路:



1. 首先通过CreateFile打开设备驱动，之后进入ioctl code的探测部分，主要探测有效的ioctl code，这里ioctlbf中采用的是在DeviceIOControl中直接用NULL来作为Input Buff和Input Buff Size，在调试中发现这样做会产生大量的空指针引用漏洞，这样就不好进入后面的fuzz过程。这里如果ioctl code无效，则直接返回，开始对下一个ioctl code进行探测，最后的记录是在一个结构体，同时用日志模式记入数据库后面会说。

2. 随后会根据-f参数选择是否0x00填充，因为常规模式是使用0x41填充缓冲区（这里也可以考虑改成随机数据填充，但这里随机数据在后面fuzz中有了），而我在对多家厂商驱动进行fuzz的时候发现，0x00填充也会产生一些问题，比如空指针引用等，ioctlbf中-f参数的功能我没太弄懂为什么要加，所以这里根据我的实战后的一些想法进行了修改。

3. 走到第二步或直接进入第三步时已经确认当前是一个有效的ioctl code，随后会对ioctl code的input buff size进

行探测，主要是探测最大的buff size和最小的buff size，这里很多驱动会对input buff size进行判断，防止溢出等情况的发生，因此这里如果返回getlasterror，那么可能就是出错了，但如果返回正确，则处理正确，这样从0-max length判断找出边界就可以了。

另外这里很有可能产生溢出（池、栈都有）

4. 至此探测完成，这里会把所有数据记入一个struct中，叫做IOCTL List，该结构体定义如下：

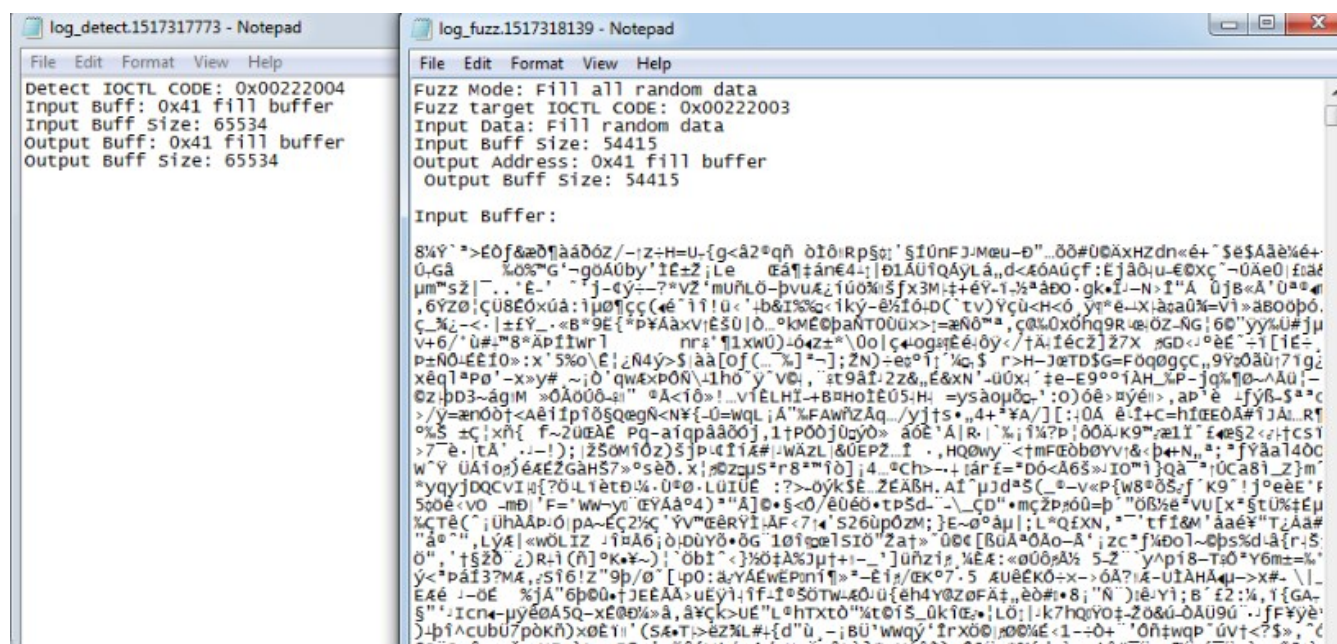
```
1 typedef struct IOCTLlist_ {  
2     DWORD IOCTL;  
3     DWORD errorCode;  
4     size_t minBufferLength;  
5     size_t maxBufferLength;  
6     struct IOCTLlist_ *previous;  
7 } IOCTLlist, *pIOCTLlist;
```

结构体以单项链表的形式保存，由于这里ioctl code也不会很多，所以单项链表足以操作不会由于查询等情况影响速



度，否则双向链表更优，这里我将ioctl code的数据通过日志模块写入数据库，以便分析查询，后续打印等工作也是根据ioctl list完成的，这是核心，随后进入Fuzz模块。

## Fuzz策略



The image shows two Notepad windows side-by-side. The left window, titled 'log\_detect.1517317773 - Notepad', contains the following text:

```
File Edit Format View Help
Detect IOCTL CODE: 0x00222004
Input Buff: 0x41 fill buffer
Input Buff Size: 65534
Output Buff: 0x41 fill buffer
Output Buff Size: 65534
```

The right window, titled 'log\_fuzz.1517318139 - Notepad', contains the following text:

```
File Edit Format View Help
Fuzz Mode: Fill all random data
Fuzz target IOCTL CODE: 0x00222003
Input Data: Fill random data
Input Buff Size: 54415
Output Address: 0x41 fill buffer
Output Buff Size: 54415

Input Buffer:
8%Y'>E0f&æ0fää0Z/-iz-H=U-{g&2*qn ö1ö:RpSg:'$f0nFJ-Meu-B"...ö0#Ü0xHZdn«é+`$e$Aäæ%é+
Ü.Gä %o"G'-goÄÜby'Ie±Z;Le æäq;än€4-;|ö1ÄÜtQÄyLä,,d<æ0Äücf:Ejädü-u-€0Xc"-ÜAe0|fæä
µm"sZ|-. "E-" "j-cy--?vZ;µuñLÖ-pvUæ;ü0%isfx3M;±+éY-1-½"äDO.gk.I-N>I"A ÜjB"A"Ü"æ"
,6YZ0;Ü8E0xüa:1µ0qçç(æ"i!ü<"b&I%ç<ikY-è%1ö-D("tv)YçÜ<H<ö Yt"e-X;äa0%V1"äB0öPö.
ç_%Z-<|±ÉY-<B"9E{"P"ÄaxviESÜ|Ö..°kmEöPaNT0ÜÜx>:æNö"ª,ç0%0x0ñq9Ræ:ÖZ-NG;60"yy%Ü#jµ
v+6/"Ü#:"8*ÄP11wr| nrs"q1xwÜ)-ö4z±*"Üo|ç4ogq;Éé:öy</tÄ;IécZ|Z7X æGD<°eE"-1[iE-;
p±N0-£E£I0>:x'5%0;£;N4Y>$ää[Of( " %]";ZN)-es°i; 'Kc,$ r>H-JæTD$G=F0q0qçc,9Y:0ÄÜ;71gZ
xëq1"Po'-x># ~;jÖ'qwæxpÖN\1h0"Y"VÖ; "æ9ÄI:2Zæ,,E&xN'-ÜÜx| "æ-E9°°fÄH_XP-jq%Ü0-ÄÄÜ;-
0z;pd3-ägim>ÖÄ000-æ" "A<i0>!"VfELHf-B#hoIEÜ5:H- =ysä0pöc-:0)öè>nyé,,ap"è -fjyB-S"æc
>/y=ænoöf<AæIip1ö5QegN<N#{-Ü=wql;A"%FAwñZÄq-/yjt±s"„4+*FA/][[:10Ä è-I+c=hiæEOÄ#1JÄL.Rf
°%S ±ç;xñ{ f~2ÜEAE Pq-atqpää00j,1tP00jÜcy0> ä0E'A|R| " %;1%?P;00A-K9"æ1I" fæS2<ç;tçS1
>7"è.ITA' "-!);(ZS0M10Z)SjP<çI1æ#-WÄZL&ÜEPZ...f ,HQöwy'<tmfæöb0YVj&<p+N,,ª:"fYäa140C
w"Y ÜA10g)éÆEZGäHS7"°sè0.x|æZçus*r8"i0j;4. "Ch>--+|äræ=-D0<A6S>IO"i}Qä"ª"iÜCa81_Z}m"
*YqyJDQCvIq{70Ü1èT0%Ü°0.LUIÜE :?>öykSÈ.ZEABH.Af"µjd°S( "°-v<P{w8°0S;f"K9"j°eèE" f
5p0e<VO -mD' f="ww-yü"æYÄÄ°4)"A]0<S<0/èÜE0°tpSd- "_çD"•mçZp;0Ü=p"0B;è°vU[x°stÜ°±Eµ
%CTè("üHÄÄp;öpa-Eç2%ç"YV"æRYI;ÄF<714'S26üp0ZM;]E-0°µj;L"QIXN, "°TfI&M"aaèx"tZ;Aa#
"ä0",L,YÄ|<wÖLIZ jipA6j;ö;DÜY0°0ç"10iæ1S10"Zat"00ç[BüÄ°0Ä0-A"jzc" fJä01-0ps<dä{r:S
ö",tS20 Z)R=1(ñ)"K*~>)"öbI<}>0tA%Jµt+- "_]uñZ1g,æEÄ:<0Ü0;A% 5-Z" y^p18-Ts0°Y6m±=æ"
ý<"päI3?Mæ,,S16!Z"9p/0 "[p0:ä;YÄEWEPn1"°-É1g/æK07.5 ÄÜEKO=x->0Ä?iä-ÜIAHÄµ->x#- \|
Eæé j-0E %jA"6p0Ü.tJEEÄÄ"uEYi;1f-I°SÖTW-æ0Ü{eh4Y0Z0FÄt,,è0#*8;"N"jè:Y1;B"É2:¼,1fGA-
S"jIcne-µyè0ASQ-xE00%ä,äçK>uE"LhTt0"°t01S_0k1æ;L0;|k7hqY0t-Z0&Ü-0ÄÜ9Ü"j fYyè-
j)p1AcubU7p0kñ)×0E1" (S&T; >èZ%L#;{d"Ü, -;BÜ"wwqY' IrX0Ü;004E<1->0+ "0ñtwqp"Üv{t<?S",ç
E0"Qu" - 30 æuæææ - æ0æ<æ"æüæ /æææ:æY"æ3ææææ00æ",æ2j;æææ00æ" - 1A&T1 æµ1A"1in" - 0BÄ
```

第一步会将一些无效地址作为input buff传入，我发现一些厂商驱动会将某些地址作为有效地址在驱动中直接引用，从而导致内存破坏的发生，这些例如某些内存高地址等等，如0xffff0000，以及低地址，如0x00001000，当然刚开始，如果用null作为input buff指针，也是无效地址。

第二步会测试某些溢出，这个实际上在探测阶段也算间接做过了，但是这里我用0x10000大小的buffer做为传入，同样input buff size大小也会很大。

第三步开始对种子进行变异，这里主要是通过随机种子填充缓冲区，因为input buff在某些驱动中可能会作为结构体，或其他成员变量，这里通过特殊的dword作为种子填充来对数据变异，这里每4个字节为一个单位依次变异，长度选择是从数据库中记录的min length到max length随机选取，这里测试过大或很小的缓冲区已经没意义，第二步已经做了。最后一步是完全随机化数据，长度是数据库记录的max length。

## Fuzzer驱动枚举模块实现

增加了驱动模块的枚举，这里具体代码实现在scan.h中，其实难度也不大。

这里有一点和驱动打开，也就是第一步息息相关，我在测试的过程中发现很多厂商的驱动打开有问题，多数都是返回errorcode 5，也就是拒绝访问，这里如果你是普通用户权限，可以尝试用administrator的权限打开，我在测试的过程中发现很多厂商都是用administrator的权限可以打开，普通用户就不行。

这样fuzz还是有意义的，但是如果需要system打开，那么

fuzz就没有意义了，毕竟系统怎么也不会给system权限，而目的就是利用ring0来EoP。

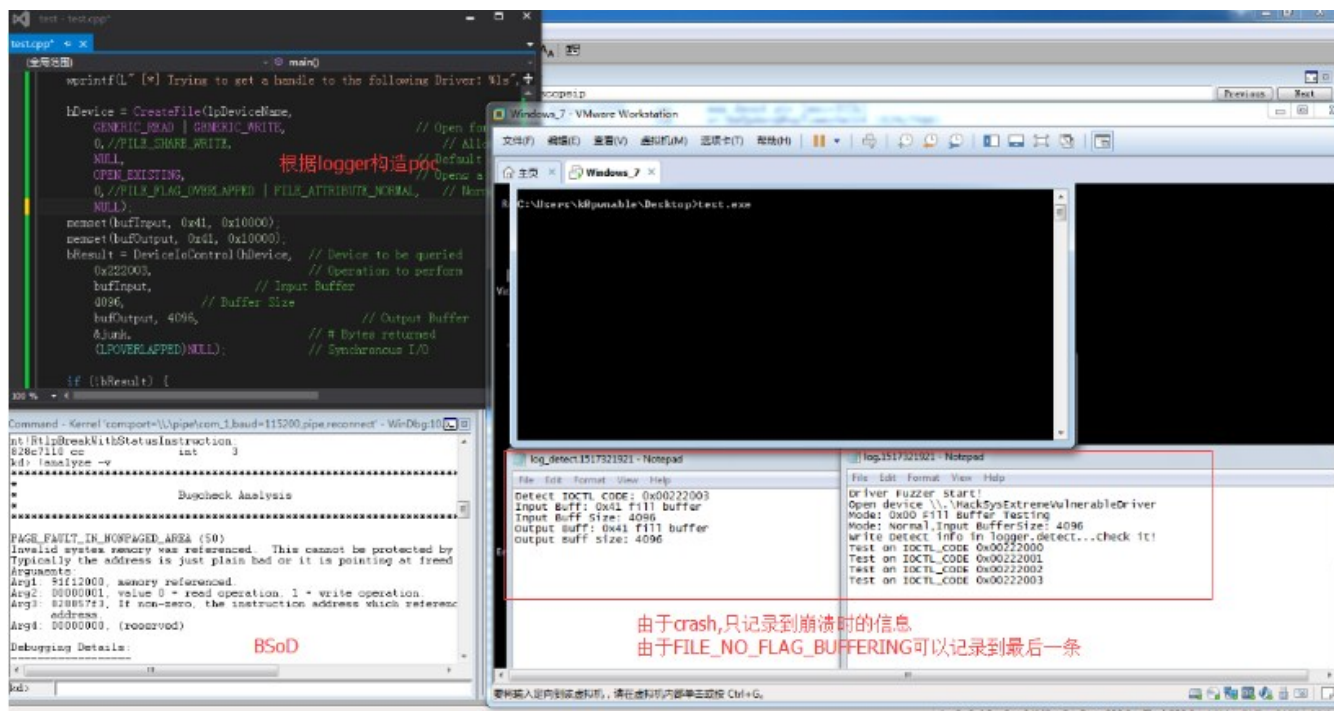
我在驱动枚举模块增加了对驱动名称的尝试打开，对可以打开或拒绝访问的驱动设备都进行了日志记录，当然，如果都不对的话也不意味着这个驱动就不存在，如果选中了某些目标，而驱动枚举模块没有找到它的话。

尝试用ida打开，分析具体的设备名称\DEVICE\XXXX，因为有些注册驱动名称并不是驱动本名，不过这个能找到打开驱动设备，因为多数驱动设备都是用的本名。

## Fuzzer 日志模块实现

ioctlibf中没有日志模块，这样产生bsod不利于还原漏洞，增加日志可以记录漏洞发生前的问题，记录传入参数和内容等等，这样能超快速写出poc。

这里一个主日志记录整个过程，探测日志和fuzz日志部分可以通过-l开启或关闭



里日志模块实现稍微费力一些，在我的logger.h中，刚开始我参考了mwr lab的kernel fuzzer中的logger，但后来大改了。

主要问题在于正常的文件写入是会先把写入内容放在一个cache里，写满了再往文件中写，这个主要好像是跟扇区对齐相关，但我们fuzz的时候会产生bsod，整个操作系统都会挂起进入异常处理，这时候缓存内容还没写到文件里，我们就记录不到发生bsod时的漏洞情况。

所以这里参考了bee13oy师傅在zer0con2017中提到的FILE\_FLAG\_NO\_BUFFERING的标记，这个标记下会将要写的内容直接写入扇区，但这里对写入的buffer有严格要求，因为扇区是要求严格对齐的，申请位置也得是对齐的。

所以我用VirtualAlloc申请512大小的空间，并且写入，这里即使输入只有几个bytes，也得申请512，查看记录中会有大量的0x00填充，但是影响不大，用常规的文本打开即可正常查看。